НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Веб-застосунок для презентацій на багатьох пристроях на архітектурі клієнт-сервер»

Виконав (-ла):
студент (-ка) IV курсу, групи ІТ-51
Левченко Тимур Ігорович

_____

Керівник:
Професор, д.т.н., проф. Шемаєв В.М.

_____

Рецензент:
Директор ТОВ «Опінов8 Україна» Приймак Г.В.

_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)
Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

ЗАВДАННЯ
на дипломний проект студенту
Левченку Тимуру Ігоровичу

1.Тема проекту «Веб-застосунок для презентацій на багатьох пристроях на архітектурі клієнт-сервер», керівник проекту професор, д.т.н., проф. Шемаєв В.М., затверджені наказом по університету від «___»_____ 2019 р. №_____

2. Термін подання студентом проекту _____

3. Вихідні дані до проекту

Операційна система Windows 7, мова програмування JavaScript, середовище програмування Visual Studio Code, цільова платформа Node.js 8, обрана для розробки технологія – React, Redux, Koa.js

4. Зміст пояснювальної записки

1. Вступ 2. Перелік скорочень, умовних позначень, термінів 3. Постановка задачі 4. Вибір та обгрунтування компонентів системи 5. Опис програмного забезпечення 6. Структура даних і ресурсів програми 7. Керівництво адміністратора

Додатки:
8. Код програми

5. Перелік графічного матеріалу

Діаграма використання, діаграма активності, діаграма розгортання, діаграма компонентів

6. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Вибір тематичного напрямку та узгодження теми курсової роботи	24.02.2019	
2	Аналіз теоретичних матеріалів та вивчення предметної області	16.04.2019	
3	Розробка технічного завдання, вибір методів та засобів реалізації задачі	20.04.2019	
4	Огляд існуючих рішень з тематики роботи	27.04.2019	
5	Розробка структури БД та проектування системи	07.05.2019	
6	Реалізація проекту	20.05.2019	
7	Налагодження та перевірка програми	22.05.2019	
8	Оформлення пояснювальної записки	27.05.2019	
9	Передзахист дипломного проекту	04.06.2019	
10	Доопрацювання пояснювальної записки та підготовка презентації	13.06.2019	
11	Захист дипломного проекту	18.06.2019	

Студент Т. І. Левченко

Керівник проекту В.М. Шемаєв

АНОТАЦІЯ

Левченко Т. І. Веб-застосунок для презентацій на багатьох пристроях на архітектурі клієнт-сервер. КПІ ім. Ігоря Сікорського, Київ, 2019.

Ключові слова: презентація, презентація на багатьох пристроях.

Основна частина документу викладена у пояснювальній записці, виконаній на 63 сторінках, та містить 17 рисунків та 1 таблицю. Також до його змісту входить 1 додаток.

Дипломний проект присвячений розробці системи синхронізованого показу презентацій на багатьох пристроях, яка надає можливість ведучому демонструвати презентації глядачам за відсутності спеціальної для цього техніки.

Серверне програмне забезпечення написане мовою програмування високого рівня JavaScript та реалізоване за допомогою фреймворку Koa.js. Графічний інтерфейс написаний мовою програмування високого рівня JavaScript та реалізований за допомогою React і Redax на основі шаблону проектування MVC.

SUMMARY

Levchenko T.I. Web app for presentations on multiple devices using client-server architecture. KPI them Igor Sikorsky, Kyiv, 2019.

Keywords: presentation, presentation on multiple devices.

The bulk of the document is set out in the explanatory note, with 63 pages, and contains 17 of drawings and 1 tables. Its content also includes 1 applications.

The diploma project is devoted to the development of a system of synchronized presentation of presentations on multiple devices, which provides the opportunity for the leader to present presentations to viewers in the absence of a special for this technique.

The server software is written in a high level JavaScript programming language and using Koa.js framework. The graphical interface is written in a high level JavaScript language and implemented with React and Redax based on the MVC design template.

Номер рядка	Формат	Позначення	Найменування	Кіл. листів	№ екз.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІТ51.110БАК.002 ПЗ	Пояснювальна записка			
6						
7	А3	ІТ51.110БАК.003 Д1	Діаграма прецедентів			
8						
9	А3	ІТ51.110БАК.004 Д2	Діаграма розгортання			
10						
11	А3	ІТ51.110БАК.005 Д3	Діаграма компонентів			
12						
13	А3	ІТ51.110БАК.006 Д4	Діаграма діяльності			
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						

Змн.	Арк.	№ докум.	Підпис	Дата	ІТ51.110БАК.001 ТП		
Розроб.		Левченко Т. І.			Веб-застосунок для презентацій на багатьох пристроях на архітектурі клієнт-сервер. Відомість технічного проекту	Літ.	Арк. 1 / Акрушів 1
Перевір.		Шемаєв В.М.					
Реценз.						КПІ ім. Ігоря Сікорського ФІОТ, гр. ІТ-51	
Н. Контр.							
Затверд.							

Пояснювальна записка
до дипломного проекту
на тему: «Веб-застосунок для презентацій на багатьох пристроях на архітектурі клієнт-сервер»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

Користувач — головний актор, будь-яка людина, що користується застосунком.

Ведучий — людина, що користується застосунком для демонстрації своєї презентації.

Глядач — людина, що користується застосунком для перегляду презентації.

Клієнт або клієнтська частина застосунку — частина застосунку, що відкривається у браузері користувача, графічний інтерфейс.

сервер або серверна частина застосунку — частина застосунку, що працює обробляє запроси з клієнтів.

сервер-ретранслятор — спеціальний сервер, який дублює команди від одного клієнту до інших, за умови, якщо вони мають однаковий інтидифікатор.

Кінцева точка серверу або роут — URL-адреса, по якій клієнт може доступитися до якоїсь функціональності серверу.

JWT — JSON Web Token — це відкритий стандарт для створення токенів доступу, на основі JSON.

JSON — JavaScript Object Notation — текстовий формат обміну даними, на основі об’єкту JavaScript.

CRUD — create, read, update, delete — 4 базові функції управління даними «створення, зчитування, зміна і видалення».

Точка входу — у веб-програмуванні так називається скрипт або файл, з якого починається робота програми.

Лінтер — статичний аналізатор мови програмування, інструмент який лише аналізує статичний вихідний код, що не скомпільований.

Поліфіл — у веб-програмуванні це код, який реалізує деякий функціонал, що не підтримується у якихось версіях веб-браузерів за замовчанням, це забезпечує більш-менш однакове відображення веб-сторінок в різних браузерах.

DOM — Document Object Model — не залежний від платформи і мови програмний інтерфейс, що дозволяє програмам і скриптам отримати доступ до вмісту HTML-, XHTML- і XML-документів, а також його змінювати вміст, структуру і оформлення таких документів.

Утиліта — допоміжна комп'ютерна програма в складі загального програмного забезпечення для виконання спеціалізованих типових задач.

Скрипт — це програма або програмний сценарій, які автоматизують деяку задачу, яку розробник робив би вручну, використовуючи інтерфейс програми.

Десктопний застосунок — комп’ютерна програма розрахована на виконання на персональному комп’ютері.

MVC — Model-View-Controller — схема поділу даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно.

Білдер — програма, що збирає весь код проекту до одного файлу за визначеними правилами.

Hot Reload або HMR (Hot Module Replacement) — у React: це процес оновлення графітного інтерфейсу по частинах, без перезавантаження сторінки.

ВСТУП

У сучасному світі обмін інформацією є найважливішою частиною повсякденного життя та процесу навчання. Для кращого засвоєння інформації її потрібно підсилювати графічними матеріалами. Саме тому презентації користуються такою популярністю серед навчальних закладів та підприємств. На даний момент існує вже багато програм які дозволяють користувачам створювати та показувати презентації. Менше з них мають зручний інтерфейс та легкі у використанні. А ще менше розраховані для використання на підприємствах або навчальних закладах. В основному такі програми націлені на показ презентацій на проекторах для невеликих груп людей. Що робити якщо група людей завелика або коли всі вони не можуть знаходитися в одному місці щоб побачити цю презентацію? Таких випадках застосовують більше проекторів, що тягне за собою більше складнощів з підключенням та налаштуванням обладнання. Або програми для онлайн спілкування з можливістю демонстрації екрану. Але такі рішення не завжди зручні для використання, наприклад відео потік сильно навантажує з'єднання з мережею, але всі мають гарний та швидкий інтернет на своїх пристроях.

Було б непогано якби існував застосунок, який дозволив користувачеві дивитися презентацію на будь-якому пристрої і показ презентації був би синхронізований з всіма іншими користувачами застосунку. Так як майже у кожної людини під рукою є сучасний смартфон, тоді такий застосунок дав би змогу усім користувачам змогу долучатися до презентації у будь-який час з будь-якого місця за умови що користувач має доступ до інтернету. Кількість схожих застосунків можна порахувати на пальцях, а безкоштовних зовсім немає.

Саме тому метою мого дипломного проекту є розробка застосунку який би давав змогу користувачам дивитися презентації на будь-яких пристроях та синхронізував показ презентацій. також доцільно щоб застосунок, окрім функціоналу синхронізованої демонстрації презентації, давав користувачеві

можливість для створення, зберігання та редагування цих презентацій. треба зауважити що гарною практикою є заощадження трафіку користувача, тому варто зробити так щоб презентації займали небагато пам'яті, а повідомлення для їх синхронізації були маленькі.

У пояснювальній записці будуть описані основні технології які використовуються для розробки даного застосунку, деталі реалізації та керівництво користувача.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Функціональні вимоги до програмного продукту

Функціональні вимоги визначають основний фронт робіт розробника, регламентують поведінку застосунку в тій чи іншій ситуації. Вони ставлять завдання, які повинна виконувати система.

Основною функцією даного застосунку є синхронізований показ презентацій на багатьох пристроях. Це означає що користувач може демонструвати презентацію на одному або декількох інших пристроях, засобами передачі команд управління та показом презентацій на запущених та підключених до мережі інших застосунках з найменшою можливою затримкою. Застосунок повинен працювати на пристроях під управлінням наступних систем:

— Windows

— Linux

— MacOS

— Android

— ISO

Важливою частиною функціоналу даного застосунку є можливість створення редагування та обміну презентаціями. Ця функціональність складається з декількох основних підсистем, що підтримують даний продукт. Додатковою функцією можна визначити інструменти для модифікації презентацій - комплекс функцій, які вбудовуються в застосунок і допомагають користувачеві у створенні матеріалу.

Також повинні бути доступні функції облікового запису: користувач може зареєструватися у застосунку, увійти під своїм обліковим записом та вийти з нього. Непоганою функцію була б можливість зареєструватися з використанням сторонніх облікових записів, таких як: google, facebook або інші соціальні мережі.

Якщо користувач зареєструвався у застосунку то він отримує можливість зберігати презентації у застосунку та мати доступ до них з будь-якого місця за наявності підключення до інтернету. Для ідентифікування користувача було б доцільно зробити можливість редагування публічної інформацію його облікового запису. Наявність облікового запису надасть користувачеві інструменти для створення та налаштування презентацій і доступу до них. Налаштування презентацій є функціональністю окремої частини застосунку, що передбачає створення кімнат, які представляють групу людей, які братимуть участь у презентації або її перегляді. Також повинен бути присутній режим публічного глядача, який дозволяє брати участь у відкритій презентації без реєстрації.

Далі буде розглянуто реєстрацію та авторизацію — основна функція яких це створення та редагування облікового запису, або публічної інформації користувача. Дані та презентації користувача зберігаються в базі даних. Авторизований доступ з використанням зареєстрованого облікового запису надає доступ до повної функціональності застосунку та забезпечує захищеність особистих даних користувача.

Будь-який користувач, підключений до застосунку та використовуючи будь-яку функцію, підключається за допомогою режиму сеансу, який базується на токені, який несе в собі інформацію про користувача та його роль. За наявністю облікового запису та ролі користувача можна визначити такі функції:

- незареєстрований глядач може переглядати лише відкриті презентації, які не захищені та мають загальний доступ;
- зареєстрований глядач має обліковий запис і може мати доступ до функціональних можливостей застосунку, але в межах конкретної презентації дозволяється лише дивитися;
- зареєстрований ведучий створює, модифікує презентацію, визначає доступ до презентації та особисто демонструє презентацію.

Обліковий запис необхідний для ідентифікації та авторизації користувача. Запис користувача містить всю інформацію, яку користувач може вимагати. У

поєднанні з функціоналом авторизації та реєстрації дозволяє застосунку виконувати наступні функції:

— створення, редагування та видалення облікових записів;
— визначення або підтвердження рівня доступу до функціоналу застосунку;
— зберігання інформації яку надав користувач та її повернення у разі необхідності.

Користувач може створювати кімнати та налаштовувати рівень доступу. Далі буде розглянуто функціонал кімнат, яким має надавати застосунок, відносно рівня доступу авторизованого або не авторизованого користувача:

— користувач може створювати загально доступні кімнати і будь-хто може підключитися до неї;
— користувач може створювати захищені кімнати - можуть підключитися тільки ті хто знає пароль;
— презентація захищена, і до неї можуть входити лише запрошені користувачі.

Застосунок надає функції перегляду презентації - користувач може перемикати слайди у двох напрямках. Під час демонстрації презентації у кімнаті функціонал змінюється:

— презентація синхронізована і доступний лише основний канал перегляду, кожен глядач буде повернутий до слайду який показує ведучий;
— презентації є загальними дляця глядачів, але немає потоку контролю, кожен глядач визначає послідовність перегляду презентації відповідно до своїх потреб.

Функціональність редактору надає інструменти для зручного створення та модифікації презентацій. Він містить різні функції з зручним інтерфейсом. Редактор не тільки допомагає керувати контентом, але також може налаштувати

параметри послідовності перегляду презентацій. Ця частина застосунку повністю визначає, як буде виглядати презентація.

Найважливішою функцією застосунку є трансляція дій ведучого до глядачів. Дана підсистема буде передавати інформацію про події, які відбуваються в презентації ведучого, кожному глядачеві, підключеному до кімнати, через сервер ретранслятор. Він несе відповідальність за доставку контенту у визначений час.

1.2 Нефункціональні вимоги до програмного продукту

Нефункціональні вимоги, визначають вимоги до системи в цілому, а не в окремих випадках використання, описують властивості і характеристики, які повинна демонструвати система, а також обмеження, які повинні бути дотримані в окремих випадках. Виділено такі нефункціональні вимоги:

- мультиплатформеність — застосунок повинен запускатися на будь-якому пристрої, під управлінням будь якої операційної системи, за умови, що на пристрої встановлено необхідні для роботи застосунку програми та застосунку видані необхідні права у системі;
- продуктивність — застосунок повинен виконувати всі передбачені операції без довгих очікувань і зависань. Виконання кожної операції має проходити в найкоротший термін;
- портативність — застосунок повинен запускатися з будь-якого пристрої збереження даних, за умови, що на цільовому пристрої встановлено необхідні для роботи застосунку програми та застосунку видані необхідні права у системі;
- легкість — застосунок не повинен витрачати зайвий трафік користувача під час роботи;

— експлуатаційна придатність — застосунок повинен виконувати всі необхідні вимоги і бути працездатним незалежно від часу його використання.

1.3 Огляд та аналіз аналогів розроблюваного продукту

Було знайдено та розглянуто наступні схожі рішення:

— Google Presentations;

— Microsoft PowerPoint;

— Prezi Business.

Google Presentations — веб застосунок для створення презентацій. Є вільним для публічного користування і має покращену підтримку для бізнесу. Для доступу до всіх функцій мінімально потрібний основний обліковий запис Google. Надає безліч інструментів для редагування вмісту. Дозволяє зберігати файли на одному обліковому записі за допомогою власної хмарної служби. Основною перевагою є можливість одночасної модифікації презентації декількома користувачами. Основним недоліком є відсутність вбудованої онлайнової презентації: кожен, хто має доступ, може переглядати його для себе, але відсутність режиму групового сеансу змушує використовувати службу Google Hangouts як інструмент для комунікації та презентації екрану.

PowerPoint — послуга, розроблена Microsoft як продовження попередньої версії програмного забезпечення PowerPoint. Це система, що поєднує звичайний редактор презентацій, онлайн веб-редактор і хмарні сховища. Разом ці частини дозволяють користувачеві створювати презентації та ділитися ними. Редактор також має безліч інструментів для редагування презентацій. Програмне забезпечення є ліцензованим і є доречним як корпоративне рішення. Можливість спільного використання презентацій дозволяє відкривати та редагувати його за допомогою того ж самого програмного забезпечення, але сеанс із синхронізованим переглядом не доступний без використання будь-яких

додаткових рішень, таких як Skype команди Microsoft або програмного забезпечення третьої сторони.

Prezi — це платформа для створення інтерактивних презентацій. Вона має базову безкоштовну ліцензію з великою кількістю обмежень і декількома варіантами ліцензій з вартістю, що робить її корпоративним рішенням. Для кожного облікового запису є сховище для зберігання файлів користувача. Основною особливістю є структура презентації – вона не лінійна і залежить від обраного дизайнером дерева карти. Презентації можна переглядати одночасно, користувачі можуть впливати на процес презентації та вирішувати, який вміст переглядати. Проте, презентаційний проект не може бути спільним для модифікації декількох користувачів у той самий момент часу. Система допускає лише обмежену кількість доступних тем і їх кількість залежить від ліцензії. Управління користувачем дозволяє досягти кращого рівня інтерактивності, але при великій кількості користувачів перетворення презентації в безлад. Тому така система краще для особистого інтерактивного навчання, а не колективних презентацій.

Було проаналізовано схожі рішення, їх переваги і недоліки. Всі ці дані допомогли визначити ключові особливості, які будуть корисні для кінцевого користувача. Основною функціональністю, яку має кожна система презентацій, є сховище файлів, підключене до облікового запису користувача. Це дозволить користувачеві отримувати доступ до даних в Інтернеті і змінювати його в будь-який час. Ще однією важливою функцією, яку повинна мати система презентацій, є синхронізована демонстрація контенту для кожного підключеного переглядача. Це найбільш необхідна функція, яка може допомогти людям у різних місцях отримати таку ж інформацію, як якщо б вони були разом на звичайній презентації. Інша корисна функція – не лінійний зміст презентацій. Він не буде інтерактивним, але матиме складну структуру, щоб мати можливість розділити на підтеми та організувати матеріал ефективно. Це може допомогти як ведучому, так і глядачам. Додатковим інструментом є мапа презентації – спеціальна функція дозволить

переглядати весь контент разом зі структурою і буде допомагати з орієнтацією в матеріалі.

Висновки до розділу

У цьому розділі було визначено функціональні та нефункціональні вимоги. Також було знайдено та розглянуто існуючі схожі рішення, на основі проведеного аналізу цих рішень було виокремлено їх переваги та недоліки. Використовуючи отримані данні було змінено функціональні та нефункціональні вимоги так, щоб вдосконалити розроблюваний застосунок.

Короткий перелік основних функціональних вимог до застосунку:

— реєстрація та авторизація користувача в системі;

— управління даними користувача;

— створення кімнат та підключення до них;

— синхронізація презентацій ведучого та глядачів;

— перегляд та демонстрація презентацій.

Короткий перелік основних нефункціональних вимог до застосунку:

— мультиплатформенність;

— легкість (невибагливість до трафіку);

— експлуатаційна придатність.

Основні покращення на основі проведеного аналізу:

— зберігання презентацій у хмарі;

— нелінійність презентацій;

— редактор презентацій.

2 ВИБІР ТА ОБГРУНТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ

2.1 Цільова платформа клієнтського застосунку

Застосунок має бути мультиплатформенним та працювати на багатьох пристроях під управлінням різних операційних систем. Як було визначено у функціональних вимогах, застосунок повинен працювати на визначених операційних системах:

— Windows;

— Linux;

— MacOS;

— Android;

— ISO.

Розробляти клієнтські частини застосунку для кожної з цих систем доволі важко, необхідний варіант, щоб розроблений застосунок працював одразу на всіх системах. Використання jvm або .NET Framework не варіант, так як мобільні пристрої їх не підтримують. Тому було прийнято рішення розробляти клієнтський застосунок, як веб-застосунок. Таке рішення має низку переваг перед стандартними десктопними та мобільними застосунками:

— працює в сучасних браузерах, так як браузери розроблені під кожну із перелічених операційних систем, застосунок буде працювати під кожною із цих систем;

— легкість розробки веб-застосунку в порівнянні з розробкою десктопноних та мобільних застосунків;

— не займає пам'яті на цільовому пристрої;

— завжди має майже однаковий дизайн для кожної із платформ.

Але таке рішення також має і негативні сторони:

— втрачає зайвий трафік при завантаженні застосунку браузер;

— час завантаження веб-застосунку більший за час завантаження аналогічних десктопних на мобільних застосунків.

Проаналізувавши усі позитивні та негативні сторони розробки веб-застосунку, було прийнято рішення, що позитивних сторін більше ніж негативних та розробка клієнтської частини застосунку буде вестися на веб-платформі.

Далі необхідно вибрати технологію, за допомогою якої буде розроблюватись клієнтський застосунок. Так як браузер добре підтримує js код, то всі технології базуються на ньому. Було вибрано декілька популярних фреймворків та далі їх буде порівняно:

— AngularJS;

— VueJS;

— ReactJS.

AngularJS [1] — це веб-фреймворк з відкритим вихідним кодом, підтримується Google. AngularJS — це структурна основа для динамічних веб-застосунків. Це дозволяє використовувати HTML як мову шаблону і дозволяє розширити синтаксис HTML, щоб лаконічно реалізувати компоненти застосунку. Зручна прив'язка даних до сторінки та зручне використання залежностей усувають більшу частину коду, який необхідно писати вручну. І все це відбувається в браузері, що робить його ідеальним партнером з будь-якою серверною технологією. Загальні особливості AngularJS такі:

— веб-фреймворк з відкритим вихідним кодом, розроблений Google;

— архітектура mvc;

— змінюйте dom безпосередньо;

— два шляхи прив'язки даних.

Але AngularJS має також неприємні особливості:

— обмеження в спостерігачах (не більше 2000);

— не розроблено для мобільних пристроїв;

— кілька способів зробити те ж саме, важко для розробника вибрати оптимальне рішення;

— двосторонній зв'язок перевіряє всі змінні двічі під час оновлення інтерфейсу, що робить його повільним.

VueJS [2] — це прогресивний JavaScript- фреймворк з відкритим вихідним кодом, який використовується для розробки інтерактивних веб-інтерфейсів. Це одна з найвідоміших технологій, що використовуються для спрощення веб-розробки. VueJS фокусується на інтерфейсі. Він може бути легко інтегрований у великі проекти для розробки інтерфейсу без будь-яких проблем. Початок роботи з VueJS дуже простий. Будь-який розробник може легко зрозуміти і побудувати інтерактивні веб-інтерфейси за невеликий інтервал часу. VueJS створений екс-співробітником Google. Має такі самі переваги як і AngularJS, а із недоліків:

— виникають складнощі при роботі із специфічними функціями;

— невелика кількість спільноти для підтримки проекту.

ReactJS [3] — потужний, ефективний та гнучкий фреймворк від Фейсбука з відкритим вихідним кодом для створення інтерфейсів користувача, збудований на принципі часткового оновлення веб-сторінки. Має найкращі показники серед аналогів (AngularJS, VueJS) по швидкодії та оптимізації. Механізм повторного використання компонентів дозволяє заощаджувати час при розробці та не дублювати однакову логіку у декількох місцях. Основні плюси React.js:

— віртуальний dom;

— одностороння прив’язна даних;

— легкість у використані;

— швидкодія;

— велика підтримка спільноти.

Також великим плюсом є те що React з плагіном webpack збирає весь код до одного файлу і на виході утворюється файл, який включає в себе всю реалізацію інтерфейсу застосунку і готовий до розгортання.

Тому для реалізації інтерфейсу була обрана технологія — React.js.

2.2 Цільова платформа серверного застосунку

Дослідивши популярні у даний час технології, було виокремлено три технології для реалізації серверу застосунку:

— Spring (Java);

— .NET Core (C#);

— Node.js (JavaScript).

Далі буде розглянуто ці технології детальніше.

Spring [4] — фреймворк, що забезпечує комплексну модель розробки і конфігурації для сучасних бізнес-застосунків – на будь-яких платформах. Ключовий елемент Spring — підтримка інфраструктури на рівні застосунку, тому розробники можуть зосередитися на бізнес-логіці без зайвих налаштувань в залежності від середовища виконання. Всі програми Spring вимагають багато XML коду, що дуже втомлює розробника. Багато паралельних механізмів перешкоджають розробці (звісно, паралельні механізми корисні для виконання одного і того ж завдання різними способами, але змушує витрачати багато часу).

.NET Core [5] — це сучасна платформа з відкритим вихідним кодом, багатоплатформеною та багатоцільовою для створення сучасного, швидкого та масштабованого програмного забезпечення. Проте вона має підвищену складність порівняно з іншими платформами. Фреймворк цілком залежить від вендора: майбутній розвиток буде залежати тільки від Microsoft. Керований код буде працювати повільніше з .NET, ніж нативний код. Погано інтегрується з React.js.

Node.js [6] — це крос-платформна система з відкритим вихідним кодом, побудована на середовищі виконання JavaScript [7] для швидких і масштабованих серверних і мережевих програм. Він є асинхронним, що забезпечує легкість масштабування. Будь то браузер або сервер, код Node.js працює подібним чином і є гнучким у використанні. Архітектура, орієнтована на події, задовольняє як клієнтську, так і серверну сторону, написану на JavaScript, і таким чином процес синхронізації є швидким і впорядкованим. Node.js використовує єдиний потік,

який бере на себе всі асинхронні операції вводу-виводу. Основні дії в веб-застосунках, включаючи читання або записування в базу даних, мережні підключення або файлову систему, можуть бути швидко виконані за допомогою цього пакету.

Для полегшення розробки було обрано поєднання Node.js з React.js. Таке поєднання дає найкращі результати в швидкості та простоті розробки, оптимізації та швидкодії. Також немало важливим є наявність ресурсної бази для розробки - пакетний менеджер npm. Він дозволяє з легкістю інтегрувати у застосунок готові рішення, адже ніхто не хоче “винаходити велосипед”.

2.3 Технологічний стек

Під час розробки проекту реалізовувалися певні технології. Список використаних технологій:

— React;

— Redux;

— web-pack;

— ESLint;

— babel;

— scss;

— npm;

— koa;

— nedb;

— jwt.

Далі буде розглянуто кожну з цих технологій та описано її особливості чи переваги. Технології React, Redux та scss використовувались у створенні проекту користувацького інтерфейсу, а koa, nedb та jwt — у проекті серверу. А такі технології, як web-pack, ESLint, babel та npm використовувались при розробці обох проектів.

2.3.1 React

React — це бібліотека для створення інтерфейсів користувача. Однією з його відмінних особливостей є можливість використання JSX [8], мови програмування з близьким до HTML [9] синтаксису, який компілюється в JavaScript. Розробники можуть досягати високопродуктивних веб-застосунків за допомогою Virtual DOM [10]. За допомогою React можна створювати ізоморфні програми, які допоможуть вам позбутися неприємної ситуації, коли користувач з нетерпінням чекає, коли завантаження даних буде завершено, а на екрані комп'ютера з'явиться щось інше, ніж анімація завантаження. Створені компоненти можуть бути легко змінені та повторно використані в нових проектах. Високий відсоток повторного використання коду збільшує покриття тестами, що в свою чергу призводить до більш високого рівня контролю якості. Основні концепції React:

— елементи — це об'єкти javascript, які представляють html-елементи;

— компоненти — це елементи react — частини інтерфейсу, які містять свою структуру і функціональність;

— jsx — це техніка легкого створення елементів і компонентів react;

— virtual DOM — це дерево react елементів на javascript, яке відображається в браузері;

— односпрямований потік даних — дані йдуть до дочірніх компонентів і навпаки;

— життєвий цикл компонента — події, які викликаються під час усього життя компоненти допомагають краще контролювати поведінку компоненти.

Найголовнішим в ReactJS є технологія Virtual DOM. DOM – це об’єктна модель документа, яка будується щоразу, коли користувач заходить на будь-яку сторінку сайту. У цій об’єктній моделі у вигляді дерева об’єктів зберігається вся інформація про об’єкти, які можуть бути змінені будь-яким чином за допомогою

програмного коду (скрипту) мовою Javascript. На рисунку 2.1 зображено об’єктну модель документа, яка була збудована з коду на мові HTML у дерево об'єктів.

```
<!DOCTYPE HTML>
<html>
<head>
  <title>HTML</title>
</head>
<body>DOM
</body>
</html>
```

Дерево об'єктів:

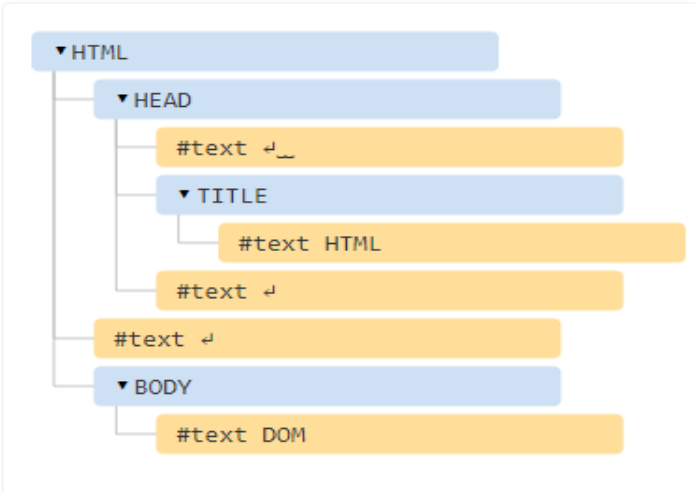

Рисунок 2.1 — Об’єктна модель документа

Virtual DOM копіює в пам’ять всю сформовану об’єктну модель документа у вигляді дерева і зберігає її. Кожного разу, коли на веб-сторінці змінюється елемент чи компонент, змінюється й DOM. Virtual DOM при зміні якогось елемента чи компонента порівнює змінений, новий DOM та збережений в пам'яті Virtual DOM і змінює лише ті елементи дерева об'єктів, які відрізняються. Таким чином досягається неймовірна швидкодія в порівнянні з аналогами. Процес порівняння та перебудови потрібних об'єктів в DOM називається процесом відновлення DOM. На рисунку 2.2 зображено графічне представлення зміни реального DOM на основі Virtual DOM.

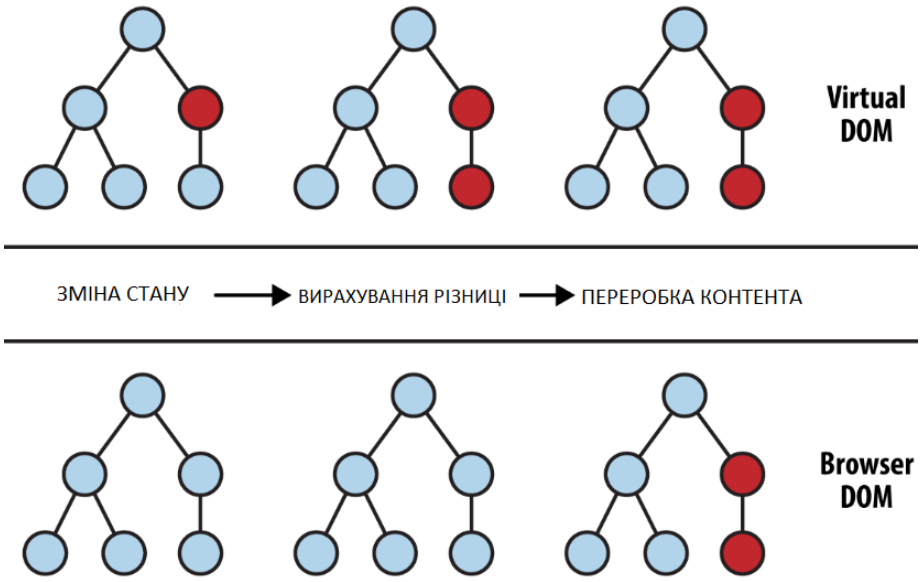

Рисунок 2.2 — Графічне представлення зміни реального DOM на основі Virtual DOM

Важливою особливість є те, що ReactJS пропагує розбиття всіх компонентів на невеликі елементи. Завдяки цьому підходу технологія Virtual DOM змінює лише ті елементи, які змінились в DOM браузера, та не змінює цілі компоненти. Також це запобігає частому дублюванню коду, це сприяє чистому та легко читаємому коду.

Однією з основних особливостей компонентів є їх стан. Ідея полягає в тому, що кожен компонент може мати свій стан та змінювати його, не впливаючи на інші компоненти.

ReactJS має чимало методів життєвого циклу компонента. Ці методи дозволяють керувати компонентом на кожному етапі його «життя». Таким чином, компоненти не є жорсткими в керуванні і дозволяють маніпулювати ними в кожний момент часу.

Оскільки ReactJS розроблено на стандарті ES6, в цій бібліотеці наявно дуже багато способів розробки асинхронних та паралельних обчислень. Це дуже корисно при запитах до сервера та отриманні інформації від нього.

2.3.2 Redux

Redux [11] — це інструмент керування станом даних та станом інтерфейсу в Javascript-застосунках. У парі React-Redux розкриває свій потенціал. Ідея Redux в тому, щоб зберігати стан компонентів в одному місці, з якого всі інші компоненти, за потреби, зможуть дізнатись про зміну стану іншого компоненту та реагувати певним чином на цю зміну. При звичайному підході React, стан компонента можна передати від батьківського компонента до нащадка або навпаки. Це називається однонаправленим потоком даних. Якщо компонентів стає все більше та ієрархія цих компонентів стає все довшою, то стає складно контролювати стан конкретного компонента в конкретний момент часу. За допомогою Redux будь-яка зміна стану конкретного компонента буде записана в глобальний стан системи і будь-який компонент буде знати про те, що певний компонент був змінений, і абсолютно неважливо, як далеко по ієрархії знаходиться цей компонент. На рисунку 2.3 зображено графічно принцип роботи Redux.

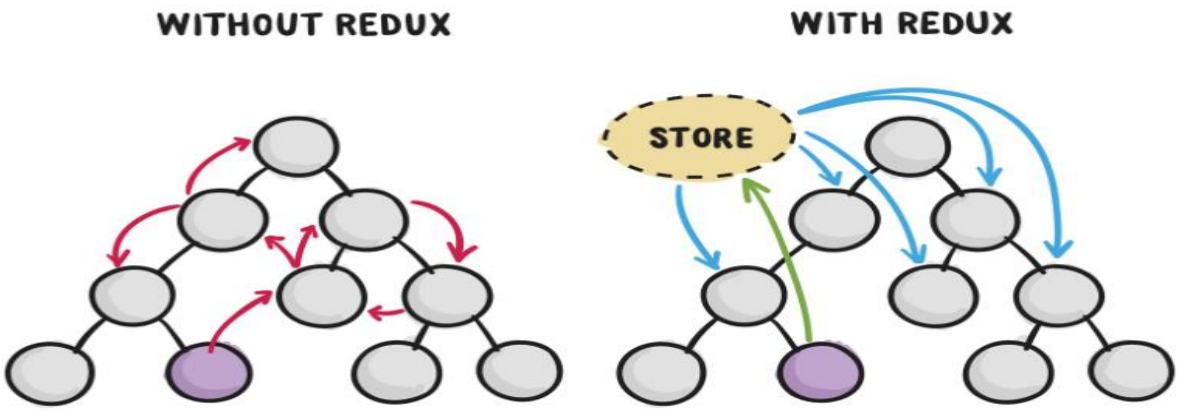

Рисунок 2.3 — Графічне представлення роботи Redux

2.3.3 Web-pack

Webpack [12] є одним з найпотужніших і гнучких інструментів для збірки користувацького інтерфейсу. Webpack — білдер з відкритим вихідним кодом, який може обробляти величезну кількість різних завдань.

Інструменти збірки стали невід'ємною частиною веб-розробки, в основному через зростання складності JS-застосунків. Білдер дозволяють нам упаковувати, компілювати, організовувати безліч ресурсів і бібліотек, необхідних для сучасного веб-проекту.

Плюси використання Webpack:

— прекрасно працює з односторінковими застосунками;

— сприймає як require () - так і import — синтакси;

— працює з стандартами es5, es6, ECMAScript;

— дозволяє здійснювати просунуте розподілення коду;

— hot reload для більш швидкої розробки за допомогою React, Vue і подібних фреймворків.

Мінуси використання Webpack:

— складно розібратися в роботі, за відсутності нормальної документації;

— робота з файлами CSS, картинками і іншими не js ресурсами для новачків збиває з пантелику.

Дуже корисним інструментом Webpack є Hot Module Replacement (HMR). HMR змінює, додає або видаляє модулі під час запуску програми без повного перезавантаження. Це може значно прискорити розробку кількома способами:

— збереження стану програми, яке втрачається під час повного перезавантаження;

— збереження часу за рахунок оновлення тільки змінених компонентів0;

— зміни, внесені в CSS / JS у вихідному коді, призводять до миттєвого оновлення браузера, яке майже можна порівняти зі зміною стилів безпосередньо в інструментах розробника.

2.3.4 ESLint

ESLint [13] є лінт утилітою з відкритим вихідним кодом. Для більшості мов програмування існують кодові лінти, і компілятори інколи включають їх в процес компіляції.

JavaScript, будучи динамічною та вільно типізованою мовою, особливо схильна до помилок розробника. Без переваг процесу компіляції, код JavaScript зазвичай виконується для того, щоб знайти синтаксичні або інші помилки. Інструменти лінтингу, такі як ESLint, дозволяють розробникам виявляти проблеми з кодом JavaScript, не виконуючи його.

Основною причиною, за якою було створено ESLint, було дозволити розробникам створювати свої власні правила. ESLint розроблений таким чином, щоб всі правила повністю підключалися. Правила за замовчуванням написані так само, як і будь-які правила плагінів. Всі вони можуть слідувати тій же схемі, як для самих правил, так і для тестів. Хоча ESLint поставляться з деякими вбудованими правилами, щоб зробити його корисним з самого початку, можливо самостійно динамічно завантажувати правила в будь-який момент часу.

ESLint написано з використанням Node.js, щоб забезпечити швидке середовище виконання та легку установку через npm.

Він призводить код до більш-менш єдиного стилю, допомагає уникнути дурних помилок, вміє автоматично виправляти багато зі знайдених проблем і відмінно інтегрується з багатьма інструментами розробки. До речі, він, як і інші лінтери, не зобов'язує до одного якогось конкретного стилю. Навпаки — можна вибрати кращу з практик і доопрацювати на свій розсуд.

2.3.5 Babel

Babel [14] — це інструмент, який в основному використовується для перетворення коду ECMAScript 2015+ у зворотну сумісну версію JavaScript у поточних і старих браузерах або середовищах. Основні речі, які Babel може робити:

— перетворити синтаксис;

— додати функції поліфіли, які відсутні у цільовому середовищі;

— перетворення вихідного коду.

Babel підтримує останню версію JavaScript через синтаксичні трансформатори. Ці плагіни дозволяють використовувати новий синтаксис прямо зараз, не чекаючи підтримки браузера. Babel може конвертувати синтаксис JSX.

Babel намагається залишатися вірним стандарту ECMAScript, наскільки це можливо. Він також може мати певні параметри, які будуть більш сумісними з специфікаціями, як компроміс із продуктивністю.

2.3.5 Scss

CSS самостійно може бути цікавим, але таблиці стилів стають більшими, складнішими і важче підтримувати. Тут може допомогти препроцесор. Sass дозволяє використовувати функції, які ще не існують у CSS, як змінні, вкладеності, успадкування та інші чудові властивості, які роблять написання CSS набагато легшим.

В Sass можна зберігати в змінних кольори, стеки шрифтів або будь-яких інших значень CSS, які потім будуть використовуватися. Sass дозволяє вкласти CSS-селектори таким же чином, як і в візуальну ієрархію HTML. Але велика кількість вкладеності робить документ менш читальним і сприйнятливим, що вважається плохої практикою.

Sass дає функціонал створювати фрагменти Sass-файлу, які будуть містити в собі невеликі шматки CSS коду, які можна використовувати в інших Sass-файлах. Це відмінний спосіб зробити CSS модульним, а також полегшити його підтримку у майбутньому. Фрагмент — це просто Sass-файл, ім'я якого починається з нижнього підкреслення. Нижнє підкреслення в імені Sass-файлу говорить компілятору про те, що це тільки фрагмент і він не повинен компілюватися в CSS. Фрагменти Sass підключаються за допомогою директив @import.

Деякі речі в CSS вельми складно писати, особливо в CSS3, де плюс до всього найчастіше потрібно використовувати велику кількість префіксів. Міксини дозволяють створювати групи декларацій CSS, які можливо використовувати по кілька разів на сайті. Навіть можна передавати змінні в міксини, щоб зробити їх більш гнучкими.

Це одна з найбільш корисних функцій Sass. Використовуючи директиву @extend можна успадковувати набори властивостей CSS від одного селектора іншому. Це дозволяє тримати Sass-файл в «чистоті». Клас-шаблон — особливий тип класів, який виводиться тільки при використанні розширення — це дозволяє тримати скомпільований CSS чистим і акуратним.

Ще однією корисною функцією є використання математику математичних операторів, таких як +, -, *, / і %.

2.3.6 Npm

Npm [15] — потужний менеджер пакетів для платформи Node JS. Він розміщує модулі на місці, щоб програмне забезпечення могло їх знайти, і розумно керувати конфліктами залежностей.

Він надзвичайно гнучкий та налаштований для підтримки різноманітних випадків використання. Найчастіше він використовується для публікації, виявлення, встановлення та розробки програм.

Npm можна налаштувати для використання будь-якого сумісного реєстру, який необхідно, і навіть запустити власний реєстр. Використання чужого реєстру може регулюватися умовами їх використання.

Npm має кілька переваг:

— заощадження простору — npm не буде встановлювати однакові залежності для різних пакетів, а буде використовувати вже встановлені:

— можливість встановлювати залежності глобально для всіх проектів з використанням npm;

— можливість інтегрувати залежності у командний рядок;

— можливість виконання скриптів;

— легка взаємодія.

2.3.7 Koa

Koa [16] — це новий веб-фреймворк, розроблений командою Express, який прагне стати меншим, виразнішим та міцнішою основою для веб-застосунків і API. Використовуючи асинхронні функції, Koa дозволяє відхиляти зворотні виклики і значно збільшувати обробку помилок. Koa не поєднує в собі жодного проміжного програмного забезпечення, і він надає елегантний набір методів, які роблять написання серверів швидким і приємним.

Koa є об'єктом, що містить масив функцій проміжного програмного забезпечення, які складаються і виконуються у вигляді, подібному до стека, за запитом. Koa схожий на багато інших систем проміжного програмного забезпечення, такі як Rack, Connect і так далі - однак ключове дизайнерське рішення було зроблено, щоб забезпечити високий рівень "цукру" на шарі проміжного програмного забезпечення низького рівня. Це покращує сумісність, надійність і робить написання проміжного програмного забезпечення набагато більш приємним.

Набір компонентів Koa, які будуть використані у розробці застосунку:

— koa — сервер Koa;

— koa-bodyparser — декодер запитів, для визначення параметрів з якими прийшов запит;

— koa2-winston-logger — логер для Koa;

— koa-response-time — автоматично відправляє негативну відповідь через деякий час, якщо не була відправлена інша.

— koa-static — дозволяє використовувати статичні файли;

— koa-router — створює маршрути для запитів;

— koa-send — відправляє відповідь на запит;

— koa-jwt — перевіряє токен авторизації;

— koa-passport — генерує токен авторизації.

2.3.8 NeDB

NeDB [17] (Node.js Embedded Database) - вбудована база даних для NodeJS, що реалізує підмножину MongoDB API. Ця легка NoSQL СУБД написана на чистому JavaScript, не має бінарних залежностей і, крім NodeJS, може використовуватися в NW.js, Electron або прямо в браузері.

NeDB забезпечує зберігання даних в простому файлі на диску в json-форматі, який схожий на колекції в MongoDB.

2.3.9 JWT

JSON Web Token [18] (JWT) — це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами, як об'єкт JSON. Всю інформацію можна перевірити та бути впевненим що вона коректна, оскільки вона підписана цифровим способом. JWT можуть бути підписані за допомогою секретного ключа (з алгоритмом HMAC) або пари

публічних / приватних ключів за допомогою RSA або ECDSA. Хоча JWT можуть бути зашифровані, щоб також забезпечити секретність між сторонами. Підписані токени можуть перевіряти цілісність даних, що містяться в ньому, тоді як зашифровані токени приховують ці претензії від інших сторін. Коли токени підписуються за допомогою пар публічних / приватних ключів, підпис також засвідчує, що тільки сторона, яка має приватний ключ, підписує її.

Нижче наведені деякі сценарії, коли корисними є веб-токени JSON:

Авторизація: це найбільш поширений сценарій використання JWT. Після того, як користувач увійшов у систему, кожен наступний запит буде включати JWT, що дозволяє користувачеві отримувати доступ до маршрутів, служб і ресурсів, які дозволяються з цим токеном.

Обмін інформацією: JSON Web Tokens — це хороший спосіб безпечної передачі інформації між сторонами. Оскільки JWT можуть бути підписані, наприклад, за допомогою пар публічних / приватних ключів, ви можете бути впевнені, що відправники є тими, кого вони вважають. Окрім того, оскільки підпис обчислюється за допомогою заголовка та корисного навантаження, можна також перевірити, що зміст не змінено.

У своїй компактній формі токени складаються з трьох частин, розділених крапками (.), серед яких є:

— заголовок;

— корисне навантаження;

— підпис.

Заголовок зазвичай складається з двох частин: типу маркера, який є JWT, і використовуваного алгоритму підписування, такого як HMAC SHA256 або RSA. Потім цей JSON кодується Base64Url для формування першої частини JWT.

Корисне навантаження є другою частиною токена. Поля несуть інформацію про користувача (ідентифікатор, ім'я, роль) і додаткові дані. Існують три типи полів: зареєстровані, публічні та приватні поля. Зареєстровані поля: це набір

попередньо визначених полів, які не є обов'язковими, але рекомендовані, щоб забезпечити набір корисних даних.

Публічні поля — поля, які можуть бути визначені за бажанням тими, хто використовує JWT.

Приватні поля — це спеціальні поля, створені для обміну інформацією між сторонами, які погоджуються на їх використання, і не є зареєстрованими або публічними даними.

Корисне навантаження — дані закодовані Base64Url, для формування другої частини JSON Web Token. Для створення частини підпису потрібно взяти закодований заголовок, закодоване корисне навантаження, секретний ключ, алгоритм, вказаний в заголовку, і підписати його. Підпис використовується для перевірки того, що повідомлення не було змінено на цьому шляху, і, у випадку токенів, підписаних приватним ключем, він також може перевірити, що відправник JWT є тим, ким він каже.

Вихідні дані складаються з трьох рядків Base64-URL, розділених точками, які можна легко передати в середовищах HTML і HTTP, при цьому вони є більш компактними в порівнянні зі стандартами на основі XML, такими як SAML.

Під час аутентифікації, коли користувач успішно увійде до системи з використанням своїх облікових даних, буде повернено JSON Web Token. Оскільки токени є обліковими даними, тому велика увага повинна бути приділина для запобігання проблем з безпекою. Загалом, не слід зберігати токени довше, ніж потрібно. Всякий раз, коли користувач хоче отримати доступ до захищеного маршруту або ресурсу, агент користувача повинен надіслати JWT, як правило, в заголовку авторизації, використовуючи схему Bearer. Захищені маршрути сервера перевірятимуть чинний JWT у заголовку авторизації, і якщо він присутній, користувачеві буде дозволено доступ до захищених ресурсів. Якщо JWT містить необхідні дані, потреба в запиті бази даних для певних операцій може бути зменшена, хоча це не завжди може бути так. Якщо токен надсилається в заголовку авторизації, спільне використання ресурсів перехресного походження (CORS) не

є проблемою, оскільки воно не використовує файли cookie. На рисунку 2.4 зображено, як JWT отримується і використовується для доступу до API або ресурсів.

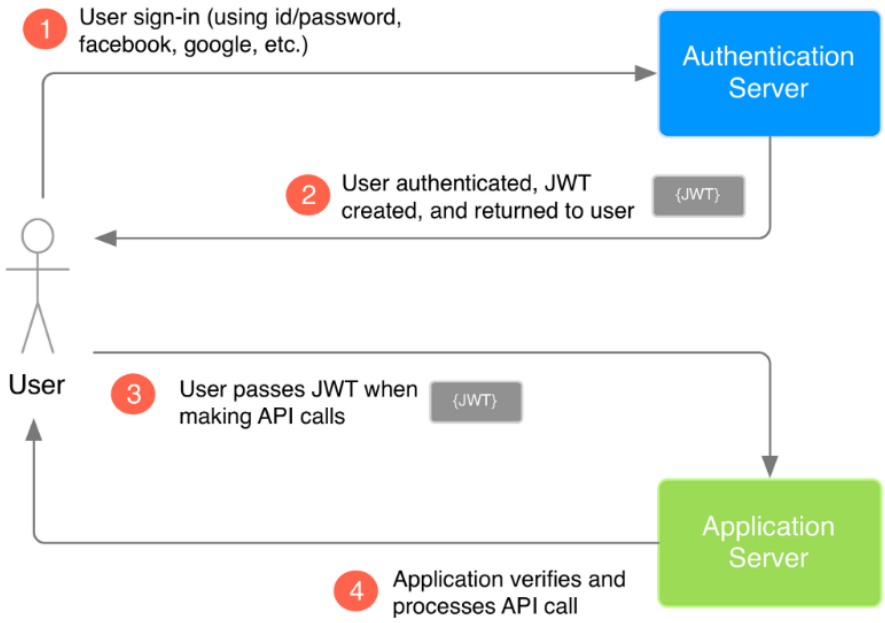

Рисунок 2.4 — Авторизація з використанням JWT

Варто зауважити, що з підписаними маркерами вся інформація, що міститься в токені, передається користувачам або іншим учасникам і вони можуть прочитати їх, навіть якщо вони не можуть його змінити. Це означає, що не потрібно вкладати секретну інформацію в токен.

Оскільки JSON менше, ніж XML, коли він кодується, його розмір також менше, що робить JWT більш компактним, ніж SAML. Це робить JWT гарним вибором для передачі в середовищах HTML і HTTP. Безпечний, SWT може бути симетрично підписаний спільним секретним ключем за допомогою алгоритму HMAC. Однак токени JWT і SAML можуть використовувати пару публічних / приватних ключів у формі сертифіката для підписання. Підписання XML цифровим підписом без введення незрозумілих дірок безпеки дуже важко порівняти з простотою підписання JSON. JSON є загальними для більшості мов програмування, оскільки вони безпосередньо відображаються на об'єктах. Навпаки, XML не має природного відображення документа в об'єкт. Це полегшує

роботу з JWT, ніж з SAML. Що стосується використання, JWT використовується в масштабі Інтернету. Це підкреслює легкість обробки на стороні клієнта JSON Web токена на декількох платформах, особливо мобільних.

Висновки до розділу

У даному розділі були порівняні та вибрані цільові платформи, на яких розроблявся застосунок. Були описані їх переваги та недоліки, на основі яких здійснювався вибір. Цільовою платформою для графічного інтерфейсу користувача (клієнтської частини застосунку) буда обрана веб-платформа та технологія React. Для розробки серверної частини застосунку — середовище Node.js. Дане поєднання цих технологій дає найкращі показники у швидкості, якості та легкості розробки.

Визначені та описані технології, які використовувались у розробці проекту чи були реалізовані. Для розробки графічного інтерфейсу застосунку були задіяні технології React, у поєднанні з Redux, та Scss. Для серверу Node.js з фреймворком Koa и базою даних NeDB. Все це використовувалось у зв’язці з ESLint та Babel, що відформатували код та зробили його сумісним з багатьма версіями браузерів та Web-pack, що мінімізував та звів всі файли до декількох файлів, які готові до завантаження на сервер.

Для стійкого захисту даних користувача, необхідно створити систему з авторизацією, щоб ніхто інший не зміг увійти в систему від імені користувача. Тому було використано технологію JSON Web Token для авторизації користувача. Технологія JWT гарантує правдивість даних закладених у токен авторизації, тому можна довіряти клієнту з валідним токеном, так як він підписується на сервері.

3 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Діаграма прецедентів

Діаграма прецедентів [19] є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами і прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

Буро побудовано діаграму прецедентів для продукту, що розробляється та описано її. Дану діаграму зображено на кресленику ІТ51.110БАК.003 Д1. Далі буде розглянуто цю діаграму по частинах, на ній зображено трьох акторів:

— користувач;

— глядач;

— ведучий.

Також на діаграмі виділено три підсистеми:

— підсистема контролю даних;

— підсистема контролю кімнат;

— підсистема показу презентації.

Кожна з яких відповідає за певні дії, які можуть виконувати актори у застосунку.

Актор користувач, зображений на рисунку 3.1 є узагальненням для акторів ведучий та глядач. Таким чином актори ведучий та глядачів можуть виконувати такі самі дії як і актор користувач, а саме авторизуватись та реєструватись у застосунку. Прецеденти реєстрація та авторизація знаходяться у підсистемі контролю даних, так як відбувається взаємодія з базою даних. Під час реєстрації особисті дані користувача записується до бази даних, а при авторизації зчитується та проходять верифікацію.

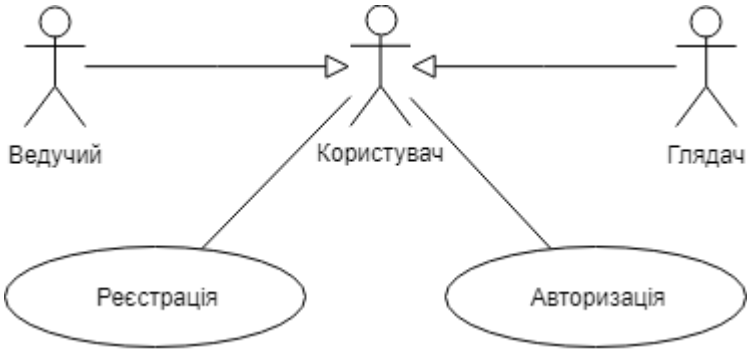

Рисунок 3.1 — Діаграма прецедентів користувача

Далі буде розглянуто рисунок 3.2 які дії може виконувати глядач у даному застосунку. Глядачу доступні такі дії з різних підсистем:

— перегляд списку активних кімнат;

— перегляд публічних даних ведучого;

— підключення до кімнати;

— перегляд презентації.

Підсистема контролю даних дозволяє глядачу продивлятися інформацію про ведучого, який створив кімнату та якого презентацію він збирається дивитися. Також підсистема контролю даних дає змогу глядачу продивлятись весь список кімнат активних на даний момент часу. Підсистема контролю кімнат деє глядачу можливість підключатися до кімнати з презентацією і, як видно з діаграми прецедентів, глядач також може вписати пароль, тим самим підключитись до захищеної кімнати. Та підсистема показу презентацій дає глядачу найважливішу

функцію — перегляд презентації. Даний випадок використання може включати в себе перемикання слайдів або перегляд мапи презентації.

Рисунок 3.2 — Діаграма прецедентів глядача

Далі буде розглянуто, які дії може виконувати ведучий у даному застосунку, зображених на рисунку 3.3. Ведучому доступні такі дії з різних підсистем:

— редагування власних публічних даних;

— робота з презентаціями;

— управління кімнатою;

— демонстрація презентації.

Рисунок 3.3 — Діаграма прецедентів ведучого

Спочатку буде розглянуто підсистему контролю даних, зображену на рисунку 3.4. Як видно з діаграми ведучий може редагувати власні публічні дані, які може переглядати глядач перед початком презентації, та працювати з своїми презентаціями. Випадок роботи з презентаціями узагальнює декілька інших дій:

— перегляд списку власних презентацій;

— редагування презентації;

— створення презентації;

— видалення презентації.

Рисунок 3.4 — Розгорнута діаграма прецеденту «Робота з презентаціями»

Ведучий може продивлятись весь список своїх презентацій, які знаходяться у базі даних застосунку. Далі ведучий може вибрати із списку якусь із презентацій та редагувати її з можливістю подальшого збереження її у базі даних застосунку. Також ведучий може видалити презентацію з списку, якщо захоче. Ведучий також може створити нову презентацію та зберегти її у застосунку. У діаграмі в застосунку показано що дія створення презентації наслідує функціонал дій завантаження файлу з презентацією та вибір збереженої у застосунку презентації,

що означає, що глядач може завантажити файл з презентацією, якщо треба відредагувати його, а вже потім зберегти презентацію.

Надалі буде розглянуто, які дії дозволяє виконувати ведучому підсистема контролю кімнат, зображені на рисунку 3.5. Як видно з рисунку дія управління кімнатою узагальнює дії:

— створення кімнати з презентацією

— налаштування кімнати з презентацією

— видалення кімнати з презентацією

Рисунок 3.5 — Розгорнута діаграма прецеденту «Створення кімнати»

Для початку ведучий може створити кімнату з презентацією. У діаграмі зазначено що створення кімнати з презентацією влече за собою налаштування кімнати з презентацією. Дія налаштування кімнати з презентацією вимагає обов'язкового виконання операцій вибір назви кімнати та вибір презентації, та необов'язковою операцію встановлення паролю для того щоб зробити кімнату приватною. Дія вибір презентації є узагальненням наступного набору дій:

— завантаження файлу з презентацією;

— вибір збереженої у застосунку презентації;

— створення презентації.

Операція створення презентації з підсистемою контролю даних вже була описана та, як зазначалося, дозволяє ведучому створити нову презентацію. Або ведучий може вибрати уже існуючу презентацію чи завантажити її з файлу презентації. На кінець ведучий може видалити кімнату щоб прибрати кімнату з списку активних, коли завершить свою демонстрацію.

Ведучий у підсистемі показу презентацій може демонструвати свою презентацію, зображені на рисунку 3.6. Цей випадок використання вимагає обов'язкового виконання дії контролю зв'язаних презентацій, що дає змогу ведучому контролювати роботу презентацій глядача, та є узагальненням для наступних дій:

— ввімкнення вікна ведучого;

— перемикання слайдів;

— перегляд мапи презентації.

Побудовано таблицю зв’язку між функціональними вимогами та варіантами використання (FR – функціональна вимога, UC – варіант використання). Результат побудови показано у таблиці 3.7.

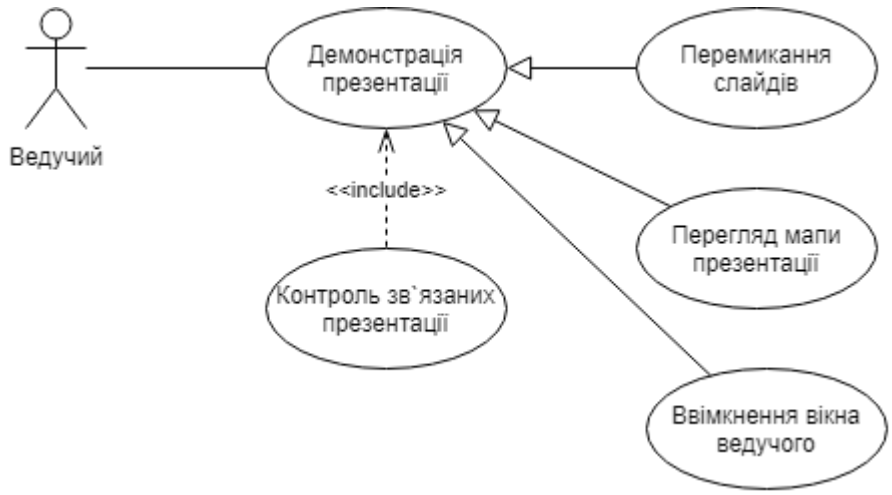

Рисунок 3.6 — Розгорнута діаграма прецеденту «Демонстрація презентації»

Таблиця 3.7 — Матриця відстеження вимог

	Реєстрація та авторизація користувача в системі	Управління даними користувача	Створення кімнат та підключення до них	Синхронізація презентацій ведучого та глядачів	Перегляд та демонстрація презентацій
Реєстрація	X				
Авторизація	X				
Перегляд списку активних кімнат		X			
Перегляд публічних даних ведучого		X			
Підключення до кімнати			X		
Перегляд презентації					X
Редагування власних публічних даних		X			
Робота з презентаціями		X			
Управління кімнатою			X		
Демонстрація презентації				X	X

Видно, що один з варіантів використання покриває декілька функціональних вимог, тому при розробці цієї активності потрібно її значно декомпозувати, адже цей варіант використання є критично важливим.

З перелічених тверджень можна дійти висновку, що система спроектована достатньо, виконані всі вимоги та не розробляються прецеденти, які не знайдуть свого застосування з точки зору логіки програмного забезпечення.

3.2 Діаграма розгортання

Застосунок має архітектуру клієнт-сервер. Клієнтська частина являє собою веб-застосунок розроблений на технології React. Серверна частина реалізована на мові програмування js на платформі Node.js. На діаграмі розгортання [20], зображеної на рисунку 3.8, можна побачити взаємодію клієнтської та серверної частин. Як видно з діаграми клієнт спілкується не тільки з основним сервером застосунку, а і з іншими клієнтами через спеціальний сервер-ретранслятор. Цей сервер ретранслятор призначений для передачі команд від одного клієнта застосунку, який являється ведучим, до інших клієнтів застосунку, які є глядачами, саме таким чином відбувається спілкування між декількома клієнтами.

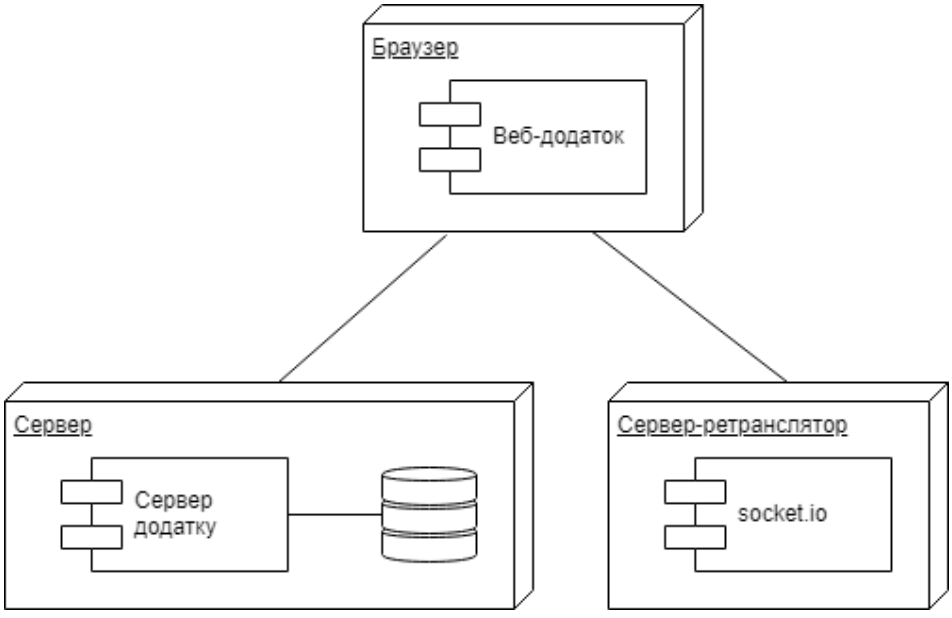

Рисунок 3.8 — Діаграма розгортання

3.3 Діаграма компонентів

Далі буде розглянуто діаграму компонентів [21], яка зображена на рисунку 3.9. Вона розбита на дві частини — клієнтську частину та серверну частину, яка свою чергу складається з наступних модулів:

— модуль маршрутизації;

— модуль авторизації;

— модуль управління даними;

— модуль контролю кімнатами;

— модуль роботи з базою даних.

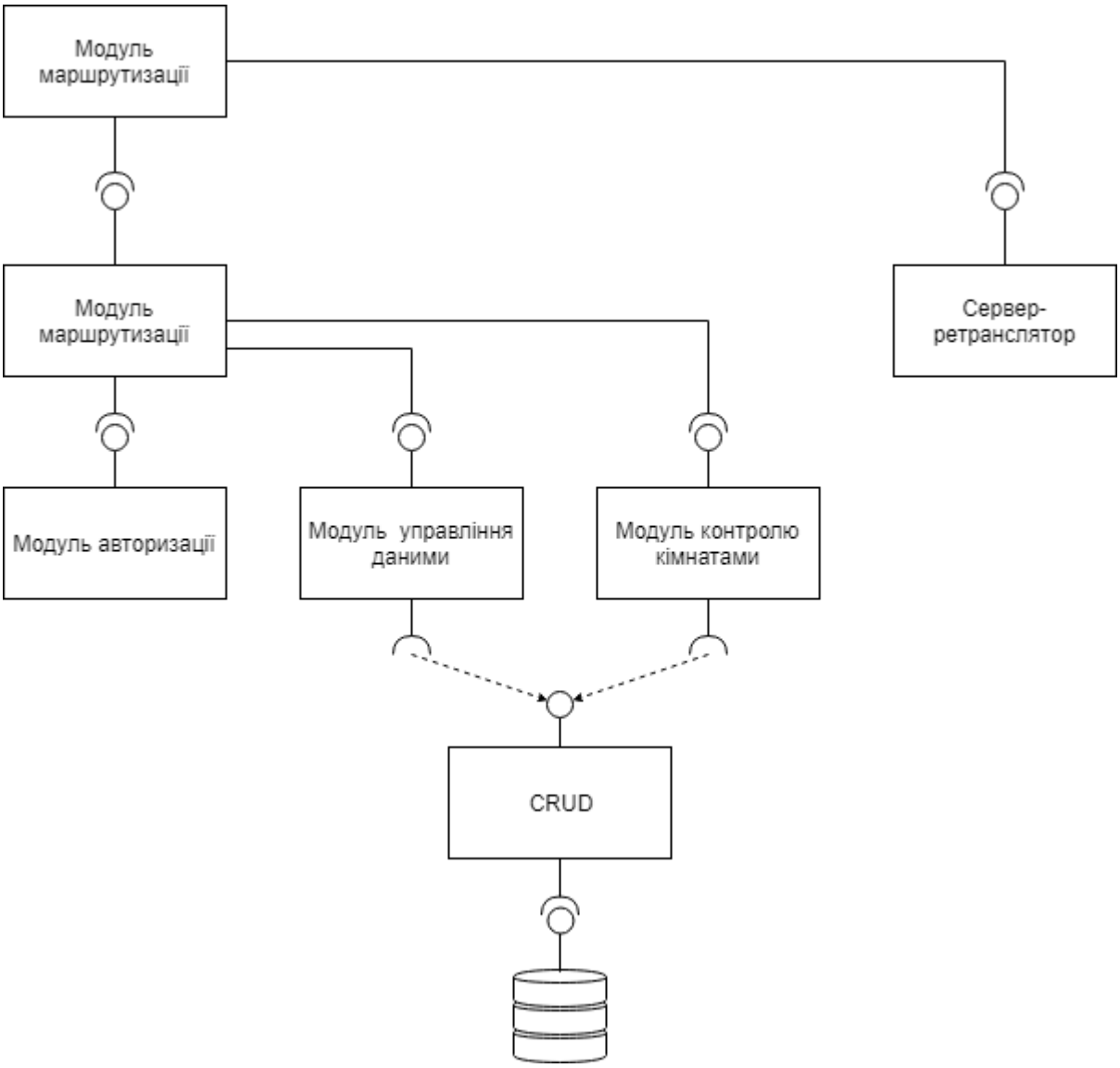

Рисунок 3.9 — Діаграма компонентів

Далі буде розглянуто призначення та функції кожного з модулів.

Модуль маршрутизація відповідає за розподілення запитів між кінцевими точками серверу та іншими модулями. Цей модуль використовує модуль авторизації як під програмне забезпечення, що дає змогу створювати приватні кінцеві точки, які можна використовувати тільки авторизованим у системі користувачам, тим які отримали необхідний токен доступу. Завдяки цьому застосунок стає більш гнучким у використанні.

Модуль авторизації відповідає за захист кінцевих точок серверу, використовуючи технологію jwt авторизації. Даний модуль перевіряє кожний запит до сервера на наявність токену та у разі його присутності перевіряє його на валідність. Якщо токену немає, або він невалідний, або його час життя вже закінчився модуль авторизації не дозволяє даному клієнту доступитися до необхідних кінцевих точок серверу та сповіщає модуль маршрутизації, що даний користувач не авторизований, а той в свою чергу перенаправить користувача на сторінку авторизації.

Модуль управління даними працює з будь-якими даними, які необхідно дістати з бази даних. Даний модуль є частково приватним, що зумовлено тим, що тільки ведучий може редагувати дані, а глядач тільки отримувати інформацію про ведучого. Цей модуль дає змогу ведучому виконувати наступні дії:

— редагування власних публічних даних;

— створення, зберігання чи редагування презентації;

— завантаження файлів презентації.

А користувачу дозволяє продивлятись дані ведучого кімнати.

Модуль контролю кімнатами дає змогу ведучому створювати та налаштувати кімнату з презентацію, а користувачу дає змогу продивлятись активні кімнати та заходити до них. Також дає можливість глядачу та ведучому підключатися до презентації.

Модуль роботи з базою даних являє собою прошарок між сервером застосунку та базою даних. Він надає інтерфейс для роботи з базою даних та може бути змінений у будь-який час для підключення застосунку до бази даних. Для модуля управління даними цей модуль надає функції роботи з таблицею користувачів та презентацій, а модуля контролю кімнат надає функції роботи з таблицею кімнат та презентації.

3.4 Діаграма діяльності

У кресленику ІТ51.110БАК.006 Д4 зображено діаграму діяльності [22] роботи даного застосунку, на прикладі наступних запитів до серверу:

— реєстрація;

— авторизація;

— редагування власних даних.

Далі буде розглянуто ту частину яка відповідає за реєстрацію та авторизацію користувача. На рисунку 3.10 зображено послідовність дій яка буде виконано у разі, якщо користувач захоче зареєструватися у даному застосунку.

Рисунок 3.10 — Діаграма активності претендента «Реєстрація»

У клієнтській частині застосунку користувач вводить свій логін та пароль і натискає кнопку реєстрації. Далі застосунок формує запит на сервер із даними, які ввів користувач. Даний запит не контролюється модулем маршрутизації, так як запит реєстрації ніколи не може бути приватним. Надалі модуль контролю даних визначає параметри з якими прийшов запит, дістає з них пароль користувача, шифрує його та записує всі отримані дані у базу даних через модуль роботи з базою даних. Надалі формує позитивну відповідь для клієнтської частини застосунку. Отримавши позитивну відповідь від серверу клієнтська частина автоматично перенаправляє користувача на сторінку авторизації.

Далі буде розглянуто процес авторизації користувача у системі зображений на рисунку 3.11.

Рисунок 3.11 — Діаграма активності претендента «Авторизація»

Користувач знаходиться на сторінці авторизації, водить свої логін та пароль і натискає кнопку авторизації. Застосунок відправляє дані на сервер. Цей запит також йде повз модулю маршрутизації, бо також не повинен бути приватним. Надалі модуль контролю даних дістає з запиту параметри з якими він прийшов та формує запит у базу даних для того, щоб знайти користувача за його логіном. Далі модуль контролю даних передає модулю роботи з базою даних сформований запит. Модуль роботи з базою даних знаходить у системі користувача за його логіном та повертає його до модуля контролю даних. Потім модуль контролю даних перевіряє хеші паролів і у разі якщо вони співпали передає дані до модуля авторизації, який на їх основі генерує токен авторизації та повертає його до модуля контролю даних, який в свою чергу генерує позитивну відповідь клієнту та включає токен авторизації до неї. Таким чином користувач отримує свій особистий токен авторизації і яким потім зможе авторизуватися у системі, навіть якщо вийде з застосунку. Отримавши токен авторизації клієнтська частина записує його до локального сховища та автоматично адресує користувача до його кабінету. У наступний раз, коли користувач відкриє веб-застосунок, останній автоматично зчитає токен авторизації з локального сховище браузеру та зразу переадресує користувача до його кабінету минаючи сторінку авторизації. У випадку якщо користувач авторизується у системі з іншого пристрою токен авторизації на даному пристрої буде не валідним, тому користувачу знову доведеться авторизуватись у системі з даного пристрою.

Далі буде розглянуто послідовність дій коли ведучий хоче відредагувати свої особисті дані. Послідовність дій для підтвердження валідності токена, зображена на рисунку 3.12. Натиснувши кнопку зберегти користувач ініціює запит від клієнтської частини застосунку на сервер з метою змінити його особисті дані. Так як дістатися до такої функціональності користувач може тільки після авторизації, то до його запиту до серверу буде включено його особистий токен авторизації. Далі цей запит потрапляє до модуля маршрутизації, який визначає що цей запит є приватним і перенаправляє його до модуля авторизації. Модуль

авторизації дістає із заголовку запиту токен авторизації та перевіряє його на валідність. Результат перевірки відправляється до модулю маршрутизації. Якщо токен був валідним модуль маршрутизації відправляє запит далі до його кінцевої точки. Якщо токен був не валідним, то модуль маршрутизації відправляє клієнтській частині застосунку відповідь про невалідний токен, що призведе до перенаправлення користувача на сторінку авторизації.

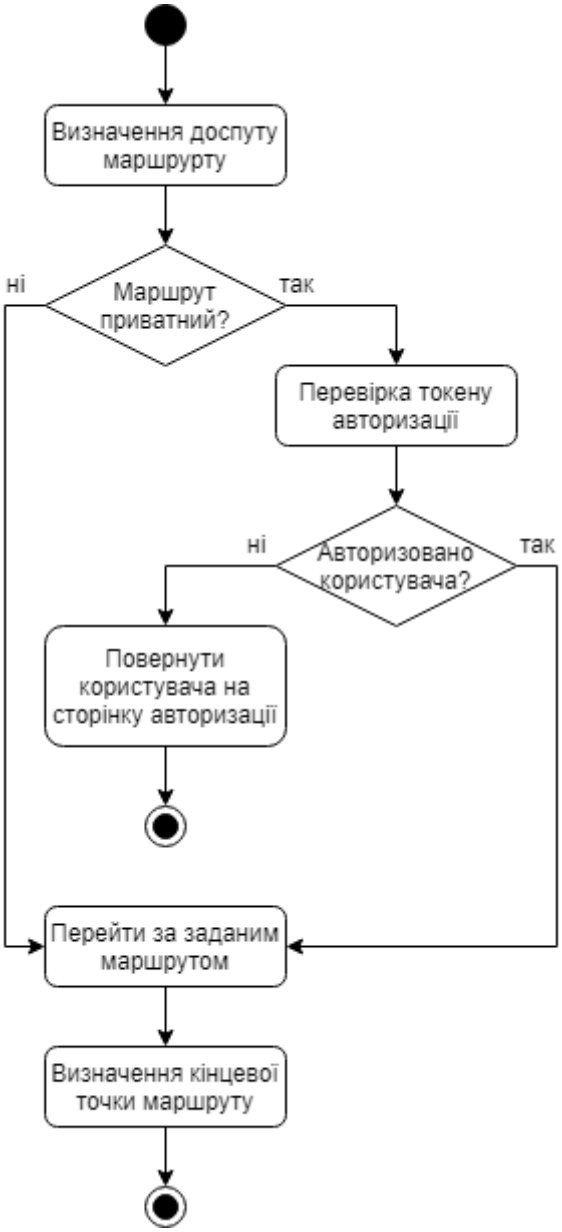

Рисунок 3.12 — Діаграма активності частини претендента «Зміна особистих даних», яка перевіряє токен авторизації

У даному разі кінцевою точкою запиту був модуль контролю даних та ціллю цього запиту була зміна особистих даних користувача. Послідовність дій викладена у рисунку 3.13. Далі запит потрапляє до модулю контролю даних, який дістає з параметрів запиту усі необхідні дані та формує запит до бази даних, який відправляє на модуль роботи з базою даних, який в свою чергу виконає його. У разі успішного виконання запиту дані користувача будуть змінені, а на клієнтську частину буде відправлена позитивна відповідь на даний запит.

Рисунок 3.13 — Діаграма активності частини претендента «Зміна особистих даних», яка відповідає за зміну даних

За схожим принципом працюють усі інші запити до серверу, тому розглядатись не будуть.

Висновки до розділу

У даному розділі були описані алгоритми роботи програмного забезпечення, зокрема алгоритм дій серверу у відповідь на наступні запити клієнту:

— реєстрація;

— авторизація;

— зміна особистих даних.

Також були створені та описані діаграми використання, активності, розгортання та компонентів, перші дві дають представлення, як працює застосунок, останні дві описують його архітектуру.

Створено діаграму прецедентів на якій описано випадки використання застосунку ведучим та глядачем. Розглянуті дії, які кожен з користувачів може виконувати знаходячись у системі.

Створено діаграму розгортання, на якій показано як розподілено систему на частини, де вони виконуються та як взаємодіють.

Створено діаграму компонентів, яка демонструє з яких модулів складається серверна частина застосунку та їх функціонал. Були розглянуть особливості поведінки цих модулів.

Створено діаграму активності, яка визначає послідовність дій, які виконує система для досягнення певної мети. Розглянуті процеси, які відбуваються у системі, як результат на дії користувача.

4 СТРУКТУРА ДАНИХ І РЕСУРСІВ ПРОГРАМИ

4.1 Структура проекту серверної частини застосунку

Далі буде розглянуто структуру проекту серверної частини застосунку. Проект виконаний на мові js на платформі Node.js з використанням фреймворку koa.js та технологій авторизації jwt. Перелік та призначення файлів конфігурації робочий середи:

— .editorconfig — файл налаштування робочої ide на якій виконувався проект;

— .eslintignore — файл конфігурації винятків для плагіну es-lint;

— .eslintrc — файл конфігурації правил плагіну es-lint;

— .gitignore — файл конфігурації винятків для системи контролю версій;

— .prettierrc — файл конфігурації правил плагіну prettier;

— package-lock.json — технічна інформація про встановлені модулі проекту;

— package.json — перелік залежностей проекту та допоміжних скриптів.

У папці config у файлі main.config.js зберігається інформація про основні налаштування проекту, такі як:

— порт запуску проекту;

— основні параметри генерування токенів авторизації;

— налаштування логерів проекту.

Також у папці знаходиться файл налаштування web-pack плагіну. Цей плагін дозволяє збирати весь код з кожного файлу проекту та упаковувати в один або декілька виконуваних файлів, що значно спрощує розгортання серверної частини застосунку на фізичному сервері. У файлі налаштувань визначено які формати файлів оброблювати. Також можна визначити спеціальні обробники для певних

розширень файлів, скільки виконуваних файлів отримати на виході та місце куди вони будуть записані.

У папці src знаходиться основні файли проекту:

— index.js;

— api/;

— app/;

— configs/;

— routing/;

— utils/.

Файл index.js є основною точкою входу в проект. Він імпортує в себе файл із папки app. Далі буде розглянуто папку api, яка включає в себе файл з основною логікою проекту. Дані файли вміщують основний функціонал створення презентацій та авторизації користувача.

У папці app зберігається файл індекс, який включає в себе всі інші файли проекту, налаштовує сервер та запускає його. Папка configs з файлом index просто деструкторизує файл configs з кореневої папки проекту.

Папка routing вміщує в себе всю логіку маршрутизації проекту. Саме тут створюється приватні та публічні маршрути проекту. Файл індекс включає в себе усі інші файли з папки routers та використовує їх, як підпрограмне забезпечення для створених маршрутів. Структура папки routing:

— index.js;

— routers/:

— apiRoutes.js;

— authRoutes.js;

— indexRoute.js;

— profileRouters.js;

— roomRoutes.js.

Папка utils вміщує в себе допоміжні функції, які використовуються в різних місцях проекту. Наприклад утиліта auth модулем авторизації, а утиліта crud

використовується у модулі контролю даних роботи з базою даних. А утиліта response використовується модулем маршрутизації, як підпрограмне забезпечення для створення відповіді на запит по замовчуванню. Також є утиліта logger яка записує усі дії серверу до файлів. Структура папки utils:

— index.js;

— logger.js;

— system.js;

— validation.js;

— auth/:

    — passport.js;

— crud/:

    — index.js;

    — nedb.js;

— response/:

    — apiResponse.js.

4.2 Структура проекту клієнтської частини застосунку

Далі буде розглянуто структуру проекту клієнтської частини застосунку. Даний проект реалізований на мові програмування js з використанням технології React та Redux. Структура кореневої папки проекту:

— configs/ — папка з файлами для налаштування плагіну webpack;

— src/ — папка з основним кодом проекту;

— .babelrc — файл з налаштуваннями для плагіну babel;

— .editorconfig — файл налаштування робочої ide на якій виконувався проект;

— .eslintignore — файл конфігурації винятків для плагіну es-lint;

— .eslintrc — файл конфігурації правил плагіну es-lint;

— .gitignore — файл конфігурації винятків для системи контролю версій;

— .prettierrc — файл конфігурації правил плагіну prettier;

— Package-lock.json — технічна інформація про встановлені модулі проекту;

— package.json — перелік залежностей проекту та допоміжних скриптів.

Далі буде розглянуто папку src у якій зберігається основний код програми:

— index.html — основний файл html у який потім записується зібраний код проекту;

— index.js — точка входу у проект;

— actions/ — папка у якій зберігаються усі можливі події які може викликати застосунок;

— components/ — папка у якій зберігаються всі візуальні компоненти клієнтської частини застосунку;

— constants/ — папка з файлами у яких визначені усі константи які використовує застосунок;

— history/ — папка з налаштуваннями модуля історії переходів застосунку;

— reducers/ — папка с файлами у якій зберігаються відповідні реакції на події які були викликані з папки actions;

— services/ — у якій зберігаються допоміжні файли які працюють з подіями та реакціями на них;

— store/ — локальне сховище для даних;

— styles/ — файли стилів.

Далі описана структура папки components та її під папок:

— App.js;

— common/:

— PrivatePropsRouteComponent.js;

— PropsRouteComponent.js;

— PublicPropsRouteComponent.js;

— header/:

— HeaderContainerComponent.js;

— HeaderViewComponent.jslogin;

— login/:

— LoginContainerComponent.js;

— LoginViewComponent.js;

— presentation/:

— PresentationInfoComponent.js;

— registration/:

— RegistrationContainerComponent.js;

— RegistrationViewComponent.js;

— room/:

— RoomCreateComponent.js;

— RoomInfoComponent.js.

Дані файли являють собою спеціальну розмітку на мові jsx, яка потім транслюється у html та включається до головного індекс файлу. Крім розмітки у файлах компонентів описані необхідні функції та поведінка сторінки. У файлі app відбувається підключення усіх компонентів та маршрутизація між ними. Наявні приватні та публічні маршрути, які виконані засобами компонентів с папки common, але логіка їх роботи відрізняється від схожих маршрутизаторів на сервері застосунку. У даному випадку користувача пропустить до приватної сторінки, якщо у локальному сховищі є токен авторизації, в іншому випадку його буде перенаправлено на сторінку авторизації.

Далі буде розглянуто файли з стилями та їх відмінності:

— semantic-ui-theme.scss — стилі для компонентів з фреймворку semantic-ui;

— styles.scss — файл, який імпортує в себе інші файли стилів;

— _large.scss — стиль для пристроїв з великим екраном;

— _medium.scss — стиль для пристроїв з середнім розміром екрану;

— _reset.scss — файл, який вимикає стилі браузеру за замовчуванням;

— _small-default.scss — стилі для невеличких пристроїв;

— _variables.scss — змінні та константи, які визначають кольори або розміри та використовуються у інших файлах стилів.

4.3 Структура бази даних

Застосунок проектувався з використанням бази даних MongoDB, але база була замінена більш легкою та портативною базою даних — NeDB, яка має аналогічний функціонал та інтерфейс підключення. Застосунок був розроблений таким чином, щоб базу даних можливо було з легкістю замінити, якщо виникне потреба масштабувати застосунок. Для цього треба лише написати файл з функціями роботи з новою базою та замінити імпорт в index файл в папці crud.

Далі буде розглянуто структуру бази даних, в якій є три таблиці: таблиця користувачів, таблиця презентацій, таблиця активних кімнат.

Таблиця користувачів призначена для зберігання інформації про користувача та зберігання хешу пароля для його авторизації. Таблиця презентацій зберігає в собі всі дані про презентацію, її власника та даних для підключення до серверу-ретранслятора, для демонстрації на багатьох пристроях. Таблиця активних кімнат служить для виводу кімнат у яких йде демонстрація презентації, основними полями є id, пароль, якщо він був заданий та id презентації яку показують.

Висновки до розділу

У даному розділі було розглянуто структуру кожного з проектів, призначення файлів проекту та взаємодія між ними. Встановлено з якими таблицями працює застосунок та їх структуру.

5 КЕРІВНИЦТВО АДМІНІСТРАТОРА

5.1 Запуск проекту серверу

Для запуску застосунку необхідно мати налаштоване середовище та пристрій достатньої потужності. Для початку треба мати середовище Node.js версії 8 або вище та менеджер пакетів npm. Так як Node.js працює під будь-якою сумісною операційною системою, тому не важливо, яка система встановлена на сервері. Якщо на сервері встановлено операційну систему Windows, необхідно пересвідчитись, що Node.js та npm записались до системної змінної path. Це необхідно для запуску програм з консолі. Якщо встановлена будь-яка Unix операційна система, то необхідно зробити системне посилання на файл запуска Node.js. Завершивши налаштування системного середовища можна перейти до налаштування самого проекту.

Спочатку треба скопіювати репозиторій чи дістати вихідний код застосунку будь-яким іншим чином, далі необхідно зайти у папку проекту серверної частини застосунку та запустити з неї консоль. У разі використання операційної системи Windows, консоль треба запустити з правами адміністратора, але у такому разі Консоль відкриється у кореневій папці Windows та необхідно буде вручну перейти до папки з проектом. Далі необхідно виконати команду npm install для того щоб встановити усі залежності проекту. Це може зайняти багато часу, так як залежностей багато. Коли залежності були встановлені можна виконати команду npm run build для того щоб зібрати застосунок у вихідний файл. Або можна запустити команду npm start для того щоб застосунок збирався сам кожного разу, як зміниться будь-який файл проекту.

Зібравши проект будь-яким чином зазначеним до цього, у папці проекту з'явиться папка dist у якій буде знаходитися файл main.js. Для запуску цього файлу необхідно відкрити консоль у даній папці та виконати команду node main.js.

Команда запустить сервер застосунку, але він не буде працювати доки до нього не буде додано файли з веб-застосунком.

5.2 Запуск проекту користувацького інтерфейсу

Для запуску проекту користувацького інтерфейсу необхідно скопіювати репозиторій з клієнтською частиною застосунку та перейти у його папку. Для встановлення залежностей треба виконати команду npm install, встановлення модулів може зайняти деякий час. Надалі необхідно запустити Консоль та виконати команду npm run build для того щоб сформувати файли клієнтського інтерфейсу. Також можна запустити команду npm start для того, щоб запустився спеціальний сервер для розробки клієнтського інтерфейсу, який запускається на локальній машині на порту 8080. Особливістю цього серверу є те, що він динамічно змінює контент відкритого у браузері застосунку в залежності від змін у файлах проекту. Відбувається це при кожній зміні файла та сервер намагається якомога швидше відобразити ці зміни у браузері. У будь-якому разі у папці dist клієнтської частини застосунку з'явиться файл index.html, багато файлів розширенням chunk.js та файли ресурсів.

Згенеровані файли клієнтської частини застосунку необхідно скопіювати у папку public, яку необхідно створити біля файл main.js у проекті серверної частини застосунку. Тоді у серверу будуть всі необхідні файли для роботи.

У кореневі папці проекту серверної частини застосунку знаходиться папка config в якій існує файл config.main.js у якому прописуються основні налаштування серверу:

— PORT — відповідає за номер порту на якому буде працювати сервер, рекомендується використовувати номери 3000 або 8000;

— JWT_SECRET — даний рядок тексту є секретним ключем від токену авторизації, рекомендується використовувати абсолютно випадкові послідовності великих та маленьких латинських літер, цифр та

спеціальних знаків та з довжиною ключа не менше тридцяти двох символів;

— HASH_ITERATIONS — даний параметр відповідає за якість шифрування паролю користувача, не рекомендується встановлювати цифру більше двадцяти;
— ACCESS_TOKEN_TTL — даний параметр відповідає за час життя токену авторизації, визначає кількість секунд до того, як токен стане не валідним;
— SALT — додаткова послідовність символів, яка додається до паролю для забезпечення кращого шифрування, має такі самі рекомендації, як секретний ключ;
— PROD_LOG_TRANSPORTS — масив адресів файлів у файловій системі, призначені для запису в них всіх логів, різного рівня важливості, зібраного проекту;
— DEV_LOG_TRANSPORTS — інший масив адресів файлів у файловій системі, призначені для запису в них всіх логів, різного рівня важливості, проекту, що знаходиться у розробці.

Якщо запустити сервер та перейти на локальний мережевий адрес за вказаним у файлі налаштувань, портом, то відкриється застосунок.

5.3 Допоміжні команди

Далі буде зазначено перелік команд, які можуть допомогти у подальшій розробці податку, виконувати їх треба у кореневі будь-якого проекту:

— lint — перевіряє весь проект на наявність синтаксичних помилок, та відхилень від заданих правил написання коду;
— lint-fix — виправляє усі синтаксичні помилки та відхилення від заданих правил написання коду, які може виправити;

— lint:watch — перевіряє проект на синтаксичні помилки та відхилення відновлення від правил у режимі реального часу та виводить нові повідомлення та попередження у консоль;

— lint:css — перевіряє файли css на наявність синтаксичних помилок, доступно тільки для проекту клієнтської частини застосунку;

— test — виконує тести, якщо вони є, та виводить їх результати.

Висновки до розділу

У даному розділі було розглянуто, яке середовище необхідно для запуску застосунку та які додаткові програми мають бути встановлені — Node.js версії 8 або вище та менеджер пакетів npm. Також було розглянуто, як запускати або збирати проекти. Визначено таку послідовність дій, для запуску серверу:

— перейти до кореневої папки клієнтського проекту;

— виконати команду npm install;

— виконати команду npm run build;

— скопіювати вміст папки dist до dist/public серверного проекту;

— перейти до кореневої папки серверного проекту;

— виконати команду npm install;

— виконати команду npm run build;

— перейти до папки dist;

— виконати команду node main.js.

Також було розглянуто структуру файлу налаштування та додаткові команди, які допоможуть перевірити проект на помилки.

ВИСНОВКИ

Було спроєктовано та розроблено програмний застосунк для широкої аудиторії, який дозволяє синхронізовано показувати презентації на багатьох пристроях.

Під час виконання проекту було визначено важливість створення даного застосунку, його актуальність у даний час та затребуваність у навчальних закладах чи підприємствах. Було сформовано та виділено основні функціональні та не функціональні вимоги до застосунку. Далі було проведено порівняльний аналіз існуючих схожих рішень, визначені їх недоліки та переваги, на основі яких були покращені функціональні вимоги до розроблюваного застосунку.

Зважаючи на не функціональні вимоги, що було виділено, було проведено аналіз доцільності розробки програмного застосунку використовуючи певну технологією та на певні платформі. Так, як всі операційні системи й платформи мають свої особливості, було вирішено розробляти програмний веб-застосунок для браузерів. Було виділено та описано переваги такого вибору. Також було виділено та описано всі допоміжні технології, що реалізовувалися у проекті чи пришвидшували, або полегшували розробку.

Дослідження предметної області показало, яким має бути застосунок, які вимоги він має задовольняти та які функції та можливості надавати кінцевому користувачу. Таким чином було виділено акторів, які можуть користуватися застосунком та складено прецеденти використання застосунку глядачем та ведучим, визначені головні сценарії використання застосунку. На основі цього було створено діаграму прецедентів та описано її.

Спроектовано систему, яка складається з клієнту та серверу, які в свою чергу складаються з підмодулів, які взаємодіють між собою. Для опису розміщення їх та взаємодії були створені та описані діаграма розгортання та діаграма компонентів. На останній розкрито взаємодію компонентів серверу, включаючи базу даних.

На основі роботи створених модулів було створено діаграму діяльності, яка описує послідовності дії, які виконує сервер, як реакцію на дію користувача. На діаграмі діяльності зображено, що відбудеться в разі, коли користувач захоче зареєструватись, авторизуватись чи виконати якусь дію, наприклад змінити свої особисті данні.

Було розглянуто структури клієнтського та серверного проектів, описано призначення файлів. Також було розглянуто, як запускати та збирати проекти, визначено необхідне для роботи середовище. Зазначено перелік команд для запуску та допоміжних команд.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AngularJS — Superheroic JavaScript MVW Framework — Режим доступу до ресурсу: https://angularjs.org/.
2. Vue.js — Superheroic JavaScript MVW Framework — Режим доступу до ресурсу: https://vuejs.org/.
3. React – A JavaScript library for building user interfaces — Режим доступу до ресурсу: https://reactjs.org/.
4. Spring [Електронний ресурс] — Режим доступу до ресурсу: https://spring.io/.
5. .NET Core [Електронний ресурс] — Режим доступу до ресурсу: https://dotnet.microsoft.com/.
6. NodeJS [Електронний ресурс] — Режим доступу до ресурсу: https://nodejs.org/en/docs/.
7. JavaScript [Електронний ресурс] — Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
8. Introducing JSX [Електронний ресурс] — https://reactjs.org/docs/introducing-jsx.html.
9. HTML [Електронний ресурс] — Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/HTML.
10. Что такое Virtual DOM? [Електронний ресурс] — Режим доступу до ресурсу: https://habr.com/ru/post/256965/.
11. Redux — A Predictable State Container for JS Apps [Електронний ресурс] — Режим доступу до ресурсу: https://redux.js.org/.
12. webpack [Електронний ресурс] — Режим доступу до ресурсу: https://webpack.js.org/.
13. ESLint — Pluggable JavaScript linter [Електронний ресурс] — Режим доступу до ресурсу: https://eslint.org/.

14. Babel — The compiler for next generation JavaScript [Електронний ресурс] — Режим доступу до ресурсу: https://babeljs.io/.
15. npm | build amazing things [Електронний ресурс] — Режим доступу до ресурсу: https://www.npmjs.com/.
16. Koa — next generation web framework for node.js [Електронний ресурс] — Режим доступу до ресурсу: https://koajs.com/.
17. NeDB [Електронний ресурс] — Режим доступу до ресурсу: https://habr.com/ru/post/301916/.
18. JSON Web Tokens [Електронний ресурс] — Режим доступу до ресурсу: https://jwt.io/.
19. Use case diagram [Електронний ресурс] — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Use_case_diagram.
20. Deployment diagram [Електронний ресурс] — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Deployment_diagram.
21. Component diagram [Електронний ресурс] — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Component_diagram.
22. Activity diagram [Електронний ресурс] — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Activity_diagram.

ДОДАТОК А

А.1 Основний код клієнтської частини застосунку

package.json

```
{
  "name": "back-office-front",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "build": "cross-env NODE_ENV=production webpack --config
configs/webpack.prod.config.js --color -p --progress --hide-modules
--display-optimization-bailout",
    "start": "set NODE_ENV=development && node tools/server",
    "start:tunnel": "cross-env NODE_ENV=development
ENABLE_TUNNEL=true node tools/server",
    "start:production": "npm run test && npm run build && npm run
start:prod",
    "start:prod": "cross-env NODE_ENV=production node tools/server",
    "lint:eslint": "./node_modules/.bin/eslint  ./src",
    "lint:eslint:watch": "./node_modules/.bin/esw --config
./.eslintrc --color  --watch ./src",
    "lint:css": "stylelint './app/**/*.js'",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "add-asset-html-webpack-plugin": "^2.1.3",
    "babel-cli": "^6.26.0",
    "babel-core": "^6.26.3",
    "babel-eslint": "^8.2.5",
    "babel-loader": "^7.1.5",
    "babel-plugin-dynamic-import-node": "^2.0.0",
    "babel-plugin-import": "^1.8.0",
    "babel-plugin-react-intl": "^2.4.0",
    "babel-preset-env": "^1.7.0",
    "babel-preset-react": "^6.24.1",
    "babel-preset-stage-0": "^6.24.1",
    "circular-dependency-plugin": "^5.0.2",
    "css-loader": "^1.0.0",
    "eslint": "^5.0.1",
    "eslint-config-airbnb": "^17.0.0",
    "eslint-config-prettier": "^2.9.0",
```

```
    "eslint-plugin-import": "^2.13.0",
    "eslint-plugin-jsx-a11y": "^6.1.0",
    "eslint-plugin-prettier": "^2.6.2",
    "eslint-plugin-react": "^7.10.0",
    "eslint-watch": "^4.0.1",
    "express": "^4.16.3",
    "file-loader": "^1.1.11",
    "glob": "^7.1.2",
    "html-loader": "^0.5.5",
    "html-webpack-plugin": "^3.2.0",
    "image-webpack-loader": "^4.3.1",
    "less": "^3.5.3",
    "less-loader": "^4.1.0",
    "ngrok": "^3.0.1",
    "node-sass": "^4.9.1",
    "prettier": "^1.13.7",
    "sass-loader": "^7.0.3",
    "style-loader": "^0.21.0",
    "stylelint": "^9.3.0",
    "svg-url-loader": "^2.3.2",
    "url-loader": "^1.0.1",
    "webpack": "^4.15.1",
    "webpack-cli": "^3.0.8",
    "webpack-dev-middleware": "^3.1.3",
    "webpack-hot-middleware": "^2.22.2"
  },
  "dependencies": {
    "axios": "^0.18.0",
    "connected-react-router": "^4.3.0",
    "cross-env": "^5.2.0",
    "history": "^4.7.2",
    "loaders.css": "^0.1.2",
    "minimist": "^1.2.0",
    "prop-types": "^15.6.2",
    "react": "^16.4.1",
    "react-dom": "^16.4.1",
    "react-loaders": "^3.0.1",
    "react-redux": "^5.0.7",
    "react-router": "^4.3.1",
    "react-router-dom": "^4.3.1",
    "redux": "^4.0.0",
    "redux-devtools-extension": "^2.13.5",
    "redux-thunk": "^2.3.0",
    "semantic-ui-css": "^2.3.2",
    "semantic-ui-react": "^0.82.0"
  }
}
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { ConnectedRouter } from 'connected-react-router';
import { Provider } from 'react-redux';

import App from './components/App';
import './styles/styles.scss';

import configureStore from './store/configureStore';
import { history } from './history';

const store = configureStore();

const MOUNT_NODE = document.getElementById('app');

const render = () => {
  ReactDOM.render(
    <Provider store={store}>
      <ConnectedRouter history={history}>
        <App />
      </ConnectedRouter>
    </Provider>,
    MOUNT_NODE
  );
};

if (module.hot) {
  // Hot reloadable React components and translation json files
  // modules.hot.accept does not accept dynamic dependencies,
  // have to be constants at compile-time
  module.hot.accept(['./components/App.js'], () => {
    ReactDOM.unmountComponentAtNode(MOUNT_NODE);
    render();
  });
}

render();
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>Stats</title>
</head>
<body>
  <div id="app"></div>
</body>
</html>
```

alert.actions.js

```
import alertConstants from '../constants/alert.constants';

function success(message) {
  return { type: alertConstants.SUCCESS, message };
}

function error(message) {
  return { type: alertConstants.ERROR, message };
}

function clear() {
  return { type: alertConstants.CLEAR };
}

export default {
  success,
  error,
  clear
};
```

loading.actions.js

```
import loadingConstants from '../constants/loading.constants';

export function startLoading() {
  return { type: loadingConstants.START_LOADING };
}

export function finishLoading() {
  return { type: loadingConstants.FINISH_LOADING };
}
```

user.actions.js

```
import { push } from 'connected-react-router';
import userConstants from '../constants/user.constants';
```

```
import alertActions from './alert.actions';
import { login as loginService, logout as logoutService } from
'../services/userService';

export const login = (username, password) => {
  return dispatch => {
    dispatch(request({ username }));
    loginService(username, password)
      .then(user => {
        dispatch(success(user));
        dispatch(push('/cabinet'));
      })
      .catch(error => {
        // FIXME
        dispatch(failure());
        if (error.response && error.response.status === 401)

dispatch(alertActions.error(error.response.data.msg.toString()));
        else dispatch(alertActions.error('Error. Please, contact
administrator.'));
      });
  };

  function request(user) {
    return { type: userConstants.LOGIN_REQUEST, user };
  }
  function success(user) {
    return { type: userConstants.LOGIN_SUCCESS, user };
  }
  function failure(error) {
    return { type: userConstants.LOGIN_FAILURE, error };
  }
};

export const logout = () => {
  logoutService();
  return { type: userConstants.LOGOUT };
};

export const changeUser = userId => {
  return {
    type: userConstants.CHANGE_USER,
    current: userId
  };
};
```

App.js

```
import React from 'react';
import { Redirect, Route, Switch, Link } from 'react-router-dom';
import PropTypes from 'prop-types';
import { connect } from 'react-redux';

import HeaderContainerComponent from
'./header/HeaderContainerComponent';
import PrivatePropsRoute from './common/PrivatePropsRouteComponent';
import PublicPropsRoute from './common/PublicPropsRouteComponent';

import LoginContainerComponent from
'./login/LoginContainerComponent';
import RegistrationContainerComponent from
'./registration/RegistrationContainerComponent';
import RoomCreateComponent from './room/RoomCreateComponent';
import RoomInfoComponent from './room/RoomInfoComponent';
import PresentationInfoComponent from
'./presentation/PresentationInfoComponent';
```

App.js

```
import { history } from '../history';
import alertActions from '../actions/alert.actions';

const Maintatis = () => <div style={{ color: 'red' }}>this page
under maintenance</div>;

class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      isDesktop: false
    };
    const { dispatch } = this.props;
    history.listen(() => {
      dispatch(alertActions.clear());
    });
  }

  componentDidMount() {
    window.addEventListener('resize', this.isDesktop, true);
    this.isDesktop();
  }

  componentWillUnmount() {
    window.removeEventListener('resize', this.isDesktop, true);
  }
```

```
  handleMenuButtonClick = () => this.setState(prevState => ({
isSidebarOpen: !prevState.isSidebarOpen }));

  isDesktop = () => {
    if (window.matchMedia('(min-width: 1140px)').matches)
this.setState({ isDesktop: true });
    else this.setState({ isDesktop: false });
  };

  render() {
    const { isDesktop } = this.state;
    return (
      <div>
        {/* <HeaderContainerComponent
handleMenuButtonClick={this.handleMenuButtonClick} /> */}
        <Switch>
          <Route exact path="/registration" render={props =>
<RegistrationContainerComponent {...props} />} />
          <PublicPropsRoute exact path="/login"
component={LoginContainerComponent} />
          <Route exact path="/presentation" render={props =>
<Maintatis {...props} />} />
          <Route exact path="/presentation/:id" render={props =>
<PresentationInfoComponent {...props} />} />
          <PrivatePropsRoute exact path="/room"
component={RoomCreateComponent} />
          <PrivatePropsRoute exact path="/room/:id"
component={RoomInfoComponent} />
          <Redirect from="*" to="/room" />
        </Switch>
      </div>
    );
  }
}

App.propTypes = {
  dispatch: PropTypes.func.isRequired
};

const mapStateToProps = ({ auth, router }) => ({
  auth,
  router
});

export default connect(mapStateToProps)(App);
```

PrivatePropsRouteComponent.js

```
import React from 'react';
import { Route, Redirect } from 'react-router-dom';
import PropTypes from 'prop-types';

const renderMergedProps = (component, ...rest) => {
  const finalProps = Object.assign({}, ...rest);
  return React.createElement(component, finalProps);
};

const PrivatePropsRoute = ({ component, ...rest }) => (
  <Route
    {...rest}
    render={routeProps =>
      localStorage.getItem('user') ? (
        renderMergedProps(component, routeProps, rest)
      ) : (
        <Redirect to={{ pathname: '/login', state: { from:
routeProps.location } }} />
      )
    }
  />
);

PrivatePropsRoute.propTypes = {
  component: PropTypes.func.isRequired
};

export default PrivatePropsRoute;
```

PropsRouteComponent.js

```
import React from 'react';
import { Route } from 'react-router-dom';
import PropTypes from 'prop-types';

const renderMergedProps = (component, ...rest) => {
  const finalProps = Object.assign({}, ...rest);
  return React.createElement(component, finalProps);
};

const PropsRoute = ({ component, ...rest }) => (
  <Route {...rest} render={routeProps =>
renderMergedProps(component, routeProps, rest)} />
);

PropsRoute.propTypes = {
```

```
  component: PropTypes.func.isRequired
};

export default PropsRoute;
```

PublicPropsRouteComponent.js

```
import React from 'react';
import { Route, Redirect } from 'react-router-dom';
import PropTypes from 'prop-types';

const renderMergedProps = (component, ...rest) => {
  const finalProps = Object.assign({}, ...rest);
  return React.createElement(component, finalProps);
};

const PublicPropsRoute = ({ component, ...rest }) => (
  <Route
    {...rest}
    render={routeProps =>
      localStorage.getItem('user') ? (
        <Redirect to={{ pathname: '/cabinet', state: { from:
routeProps.location } }} />
      ) : (
        renderMergedProps(component, routeProps, rest)
      )
    }
  />
);

PublicPropsRoute.propTypes = {
  component: PropTypes.func.isRequired
};

export default PublicPropsRoute;
```

HeaderContainerComponent.js

```
import React from 'react';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';
import HeaderViewComponent from './HeaderViewComponent';
import { logout } from '../../actions/user.actions';

class HeaderContainerComponent extends React.Component {
  onLogoutClick = () => {
    const { dispatch } = this.props;
    dispatch(logout());
```

```
  };

  render() {
    const { handleMenuButtonClick, auth } = this.props;
    return (
      <HeaderViewComponent
        handleMenuButtonClick={handleMenuButtonClick}
        auth={auth}
        onLogoutClick={this.onLogoutClick}
      />
    );
  }
}

HeaderContainerComponent.propTypes = {
  handleMenuButtonClick: PropTypes.func.isRequired,
  dispatch: PropTypes.func.isRequired
};

const mapStateToProps = ({ auth }) => ({
  auth
});

export default connect(mapStateToProps)(HeaderContainerComponent);
```

HeaderViewComponent.js

```
import React from 'react';
import PropTypes from 'prop-types';
import { Icon } from 'semantic-ui-react';

const HeaderViewComponent = ({ handleMenuButtonClick, auth,
onLogoutClick, onLoginClick }) => {
  const { loggedIn, user } = auth;
  const shouldShow = loggedIn && user && user.at;
  // FIXME one condition for both
  return (
    <header>
      {shouldShow ? <Icon name="list"
onClick={handleMenuButtonClick} /> : null}
      {shouldShow ? (
        <div style={{ marginLeft: 'auto', display: 'flex',
alignItems: 'center' }}>
          <div
            id="user-name"
            style={{ paddingRight: '30px', color: '#ffffff',
fontSize: '1rem' }}
```

```
            onClick={() => (user.id === '0' ? onLoginClick() :
null)}>
            {user.username}
          </div>
          <Icon name="log out" onClick={onLogoutClick} />
        </div>
      ) : null}
    </header>
  );
};

HeaderViewComponent.propTypes = {
  handleMenuButtonClick: PropTypes.func.isRequired,
  auth: PropTypes.object.isRequired,
  onLogoutClick: PropTypes.func.isRequired
};

export default HeaderViewComponent;
```

LoginContainerComponent.js

```
import React from 'react';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';
import LoginViewComponent from './LoginViewComponent';
import { login } from '../../actions/user.actions';

class LoginContainerComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      username: '',
      password: ''
    };
  }

  handleSubmit = e => {
    e.preventDefault();
    const { username, password } = this.state;
    const { dispatch } = this.props;

    if (username && password) {
      dispatch(login(username, password));
    }
  };

  onUsernameChange = e => {
    this.setState({ username: e.target.value });
```

```
  };

  onPasswordChange = e => {
    this.setState({ password: e.target.value });
  };

  render() {
    const { alert } = this.props;
    return (
      <LoginViewComponent
        handleSubmit={this.handleSubmit}
        alert={alert}
        onUsernameChange={this.onUsernameChange}
        onPasswordChange={this.onPasswordChange}
      />
    );
  }
}

LoginContainerComponent.propTypes = {
  dispatch: PropTypes.func.isRequired,
  alert: PropTypes.object
};

const mapStateToProps = ({ auth: { loggingIn }, alert }) => ({
  loggingIn,
  alert
});

export default connect(mapStateToProps)(LoginContainerComponent);
```

LoginViewComponent.js

```
import React from 'react';
import PropTypes from 'prop-types';
import { Form, Button, Message, Input, Icon } from 'semantic-ui-
react';
import { Link } from 'react-router-dom';

const LoginViewComponent = ({ handleSubmit, alert, onUsernameChange,
onPasswordChange }) => {
  return (
    <div className="form-container">
      <div className="auth-title">Login</div>
      <Form>
        <Form.Field>
          <Input icon="user outline" iconPosition="left"
onChange={onUsernameChange} placeholder="Login" />
```

```
        </Form.Field>
        <Form.Field>
          <Input icon="key" iconPosition="left"
onChange={onPasswordChange} type="password" placeholder="Password"
/>
        </Form.Field>
        <Button animated="vertical" fluid type="submit"
onClick={handleSubmit}>
          <Button.Content visible>Submit</Button.Content>
          <Button.Content hidden>
            <Icon name="arrow right" />
          </Button.Content>
        </Button>
        {alert.type === 'alert-danger' ? <Message
negative>{alert.message}</Message> : null}
      </Form>
      <div className="auth-register">
        Do not have account yet? 
        <Link to="/registration">Create one</Link>
      </div>
    </div>
  );
};

LoginViewComponent.propTypes = {};

export default LoginViewComponent;
```

PresentationInfoComponent.js

```
import React from 'react';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';
import { Form, Button, Input, TextArea } from 'semantic-ui-react';
import { Redirect } from 'react-router-dom';
import axios from 'axios';

class PresentationInfoComponent extends React.Component {
  handleSubmit = e => {
    e.preventDefault();
    const { match } = this.props;
    const { id } = match.params;
    axios({
      method: 'get',
      url: `/api/public/room/${id}`
    })
      .then(({ data: { status, data } }) => (status === 'OK' ? data
: {}))
```

```
      .then(({ html }) => {
        console.log(html);
        if (html) {
          const presentationWindow = window.open(`/`, '_blank');
          presentationWindow.document.write(html);
        }
      });
  };

  render() {
    return (
      <div className="form-container">
        <Form>
          <Button fluid type="submit" onClick={this.handleSubmit}>
            Open presentation
          </Button>
        </Form>
      </div>
    );
  }
}

PresentationInfoComponent.propTypes = {};

export default PresentationInfoComponent;
```

RegistrationContainerComponent.js

```
import React from 'react';
import { connect } from 'react-redux';
import { Redirect } from 'react-router';
import PropTypes from 'prop-types';
import axios from 'axios';
import RegistrationViewComponent from './RegistrationViewComponent';

class LoginContainerComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      username: '',
      password: '',
      redirect: false
    };
  }

  handleSubmit = e => {
    e.preventDefault();
```

```
    const { username, password } = this.state;

    if (username && password) {
      axios({
        method: 'post',
        url: '/api/public/profile/create',
        data: {
          login: username,
          password
        }
      })
        .then(({ data: { status, data } }) => (status === 'OK' ?
data : {}))
        .then(data => {
          if (data === 'success') this.setState({ redirect: true });
        });
    }
  };

  onUsernameChange = e => {
    this.setState({ username: e.target.value });
  };

  onPasswordChange = e => {
    this.setState({ password: e.target.value });
  };

  render() {
    const { alert } = this.props;
    const { redirect } = this.state;
    return redirect ? (
      <Redirect to="/login" />
    ) : (
      <RegistrationViewComponent
        handleSubmit={this.handleSubmit}
        alert={alert}
        onUsernameChange={this.onUsernameChange}
        onPasswordChange={this.onPasswordChange}
      />
    );
  }
}

LoginContainerComponent.propTypes = {
  alert: PropTypes.object
};

const mapStateToProps = ({ auth: { loggingIn }, alert }) => ({
```

```
  loggingIn,
  alert
});

export default connect(mapStateToProps)(LoginContainerComponent);
```

RegistrationViewComponent.js

```
import React from 'react';
import PropTypes from 'prop-types';
import { Form, Button, Message, Input, Icon } from 'semantic-ui-
react';
import { Link } from 'react-router-dom';

const LoginViewComponent = ({ handleSubmit, alert, onUsernameChange,
onPasswordChange }) => {
  return (
    <div className="form-container">
      <div className="auth-title">Registration</div>
      <Form>
        <Form.Field>
          <Input icon="user outline" iconPosition="left"
onChange={onUsernameChange} placeholder="Login" />
        </Form.Field>
        <Form.Field>
          <Input icon="key" iconPosition="left"
onChange={onPasswordChange} type="password" placeholder="Password"
/>
        </Form.Field>
        <Button animated="vertical" fluid type="submit"
onClick={handleSubmit}>
          <Button.Content visible>Submit</Button.Content>
          <Button.Content hidden>
            <Icon name="arrow right" />
          </Button.Content>
        </Button>
        {alert.type === 'alert-danger' ? <Message
negative>{alert.message}</Message> : null}
      </Form>
      <div className="auth-register">
        Already have account? 
        <Link to="/login">Login</Link>
      </div>
    </div>
  );
};

LoginViewComponent.propTypes = {};
```

```
export default LoginViewComponent;
```

RoomCreateComponent.js

```
import React from 'react';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';
import { Form, Button, Input, TextArea } from 'semantic-ui-react';
import { Redirect } from 'react-router-dom';
import axios from 'axios';

class RoomCreateComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      presentation: '',
      redirect: false,
      id: ''
    };
  }

  onPresentationChange = e => {
    this.setState({ presentation: e.target.value });
  };

  handleSubmit = e => {
    const { at, userId, current } = this.props;

    e.preventDefault();

    const { presentation } = this.state;

    if (presentation) {
      axios({
        method: 'post',
        url: '/api/private/room/create',
        headers: {
          Authorization: `Bearer ${at}`
        },
        data: {
          presentation
        }
      })
        .then(({ data: { status, data } }) => (status === 'OK' ?
data : {}))
        .then(({ id }) => {
          if (id) this.setState({ redirect: true, id });
```

```
        });
    }
  };

  render() {
    const { redirect, id } = this.state;
    return redirect ? (
      <Redirect to={`/room/${id}`} />
    ) : (
      <div className="form-container">
        <Form>
          <Form.Field>
            <TextArea placeholder="..."
onChange={this.onPresentationChange} />
          </Form.Field>
          <Button fluid type="submit" onClick={this.handleSubmit}>
            Create room
          </Button>
        </Form>
      </div>
    );
  }
}

RoomCreateComponent.propTypes = {};

const mapStateToProps = ({
  auth: {
    user: { at, id }
  },
  user: { current }
}) => ({
  at,
  userId: id,
  current
});

export default connect(mapStateToProps)(RoomCreateComponent);
```

RoomInfoComponent.js

```
import React from 'react';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';
import { Form, Button, Input, TextArea } from 'semantic-ui-react';
import { Redirect } from 'react-router-dom';
import axios from 'axios';
```

```
class RoomInfoComponent extends React.Component {
  handleSubmit = e => {
    e.preventDefault();
    const { at, userId, current, computedMatch } = this.props;
    const { id } = computedMatch.params;
    axios({
      method: 'get',
      url: `/api/private/room/${id}`,
      headers: {
        Authorization: `Bearer ${at}`
      }
    })
      .then(({ data: { status, data } }) => (status === 'OK' ? data
: {}))
      .then(({ html }) => {
        console.log(html);
        if (html) {
          const presentationWindow = window.open(`/`, '_blank');
          presentationWindow.document.write(html);
        }
      });
  };

  render() {
    return (
      <div className="form-container">
        <Form>
          <Button fluid type="submit" onClick={this.handleSubmit}>
            Start presentation
          </Button>
        </Form>
      </div>
    );
  }
}

RoomInfoComponent.propTypes = {};

const mapStateToProps = ({
  auth: {
    user: { at, id }
  },
  user: { current }
}) => ({
  at,
  userId: id,
  current
});
```

```
export default connect(mapStateToProps)(RoomInfoComponent);
```

userService.js

```
import axios from 'axios';

export const login = (username, password) => {
  return axios({
    method: 'post',
    url: '/api/public/auth',
    data: {
      login: username,
      password
    }
  })
    .then(({ data: { status, data } }) => (status === 'OK' ? data :
{}))
    .then(data => {
      if (data.at) localStorage.setItem('user',
JSON.stringify(data));
      return data;
    });
};

export const logout = () => {
  localStorage.removeItem('user');
};

export default {
  login,
  logout
};
```

configureStore.js

```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import { composeWithDevTools } from 'redux-devtools-extension';
import { connectRouter, routerMiddleware } from 'connected-react-
router';
import rootReducer from '../reducers';
import { history } from '../history';

export default function configureStore(initialState) {
  return createStore(
    connectRouter(history)(rootReducer),
```

```
    initialState,
    composeWithDevTools(applyMiddleware(routerMiddleware(history),
thunk))
  );
}
```

A.2 Основний код серверної частини застосунку

index.js

```
/* eslint-disable global-require */
import crypto from 'crypto';
import { auth } from './configs';
import initApp from './app';
import { addUser, getUserById } from './utils/crud';

const { SALT, HASH_ITERATIONS } = auth;

(async () => {
  initApp();
})();
```

app

```
import Koa from 'koa';
import bodyParser from 'koa-bodyparser';
import koaLogger from 'koa2-winston-logger';
import responseTime from 'koa-response-time';
import cors from 'kcors';
import serve from 'koa-static';
import { log } from '../utils/logger';
import mainRouter from '../routing';
import indexRoute from '../routing/routers/indexRoute';
import { server as serverConfigs } from '../configs';

const index = new Koa();
index
  .use(responseTime())
  .use(koaLogger(log))
  .use(
    bodyParser({
      jsonLimit: '1kb',
    })
  )
  .use(cors())
  .use(mainRouter.allowedMethods())
```

```
  .use(mainRouter.routes())
  .use(serve('public'))
  .use(indexRoute)
  .on('error', err => log.error(`Error occurred:\n${err.stack}`));

export default () =>
  index.listen(serverConfigs.PORT, () => {
    log.info(`New server run at port ${serverConfigs.PORT}`);
  });
```

userController.js

```
import crypto from 'crypto';
import { auth } from '../../configs';
import { getUserById, getUserByLogin, addUser } from
'../../utils/crud';

const { SALT, HASH_ITERATIONS } = auth;

const parser = row => ({
  id: row._id,
  login: row.login,
  password: Buffer.from(row.password, 'base64'),
});

export default class User {
  static getByLogin = login =>
    new Promise((res, rej) => {
      getUserByLogin(login)
        .then(row => res(parser(row)))
        .catch(rej);
    });
  static getById = id =>
    new Promise((res, rej) => {
      getUserById(id)
        .then(row => res(parser(row)))
        .catch(rej);
    });
  static create = (login, password) =>
    new Promise((res, rej) => {
      addUser({
        login,
        password: crypto.pbkdf2Sync(password, SALT, HASH_ITERATIONS,
128, 'sha1').toString('base64'),
      })
        .then(res)
        .catch(rej);
    });
```

```
  static checkPassword = (pass, passwordHash) =>
    pass && passwordHash && crypto.pbkdf2Sync(pass, SALT,
HASH_ITERATIONS, 128, 'sha1').equals(passwordHash);
}
```

configs

```
import configs from '../../../configs/main.config';

export default configs;
export const { server } = configs;
export const { auth } = configs;
export const { log } = configs;
export const { crypt } = configs;
export const ngvx = configs.ngvxProd;
export const { DEVELOPMENT } = process.env;
```

routing

```
import jwt from 'koa-jwt';
import getRouter from 'koa-router';
import apiResponse from '../utils/response/apiResponse';
import { profilePrivateRoutes, profilePublicRoutes } from
'./routers/profileRouters';
import { roomPrivateRoutes, roomPublicRoutes } from
'./routers/roomRoutes';
import apiRoutes from './routers/apiRoutes';
import authRoutes from './routers/authRoutes';
import { DEVELOPMENT, auth as authConfigs } from '../configs';
import test from '../../backend/api/presentation/test';

const publicRouter = getRouter();
publicRouter.use('/auth', authRoutes);
publicRouter.use('/room', roomPublicRoutes);
publicRouter.use('/profile', profilePublicRoutes);

const privateRouter = getRouter();
privateRouter.use(jwt({ secret: authConfigs.JWT_SECRET, debug:
!DEVELOPMENT }));
privateRouter.use('/room', roomPrivateRoutes);
privateRouter.use('/profile', profilePrivateRoutes);

const mainRouter = getRouter();
mainRouter.use(apiResponse);
mainRouter.use('/api/public', publicRouter.routes());
mainRouter.use('/api/private', privateRouter.routes());
```

```
mainRouter.get('/ping', ctx => {
  ctx.body = 'PONG';
});

export default mainRouter;
```

auth

```
import passport from 'koa-passport';
import HTTPErrors from 'http-custom-errors';
import jwt from 'jsonwebtoken';
import { promisify } from 'bluebird';
import { Strategy as LocalStrategy } from 'passport-local';
import { Strategy as JwtStrategy, ExtractJwt } from 'passport-jwt';
import { auth as authConfigs } from '../../configs/';
import User from '../../api/users/userController';

const signAsync = promisify(jwt.sign, jwt);

const jwtPassportOptions = {
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
  secretOrKey: authConfigs.JWT_SECRET,
  jsonWebTokenOptions: {
    maxAge: authConfigs.ACCESS_TOKEN_TTL,
  },
};

const accessTokenOpts = {
  expiresIn: authConfigs.ACCESS_TOKEN_TTL,
};

passport.use(
  new LocalStrategy(
    {
      usernameField: 'login',
      passwordField: 'password',
      session: false,
    },
    (login, password, done) => {
      User.getByLogin(login)
        .then(user => {
          if (User.checkPassword(password, user.password)) {
            return done(null, user);
          }
          return done(null, false, {
            message: 'Incorrect email or password.',
```

```
          });
        })
        .catch(done);
    }
  )
);

passport.use(
  new JwtStrategy(jwtPassportOptions, (payload, done) =>
    User.getById(payload.id)
      .then(user => done(null, { id: user.id, login: user.login }))
      .catch(err => done(err))
  )
);

export const localAuthHandler = (ctx, next) =>
  passport.authenticate('local', async (err, user, info) => {
    if (user === false) {
      throw new HTTPErrors.UnauthorizedError(info.message);
    } else {
      try {
        const accessTokenPayload = {
          id: user.id,
          login: user.login,
        };
        ctx.state.result = {
          data: {
            id: user.id,
            username: user.login,
            at: await signAsync(accessTokenPayload,
authConfigs.JWT_SECRET, accessTokenOpts),
          },
        };
      } catch (e) {
        throw new HTTPErrors.UnauthorizedError('Unnable to
autorize.');
      }
    }
  })(ctx, next);

export const jwtAuthHandler = (ctx, next) =>
  passport.authenticate('jwt', async (err, user, info) => {
    if (user === false) {
      throw new HTTPErrors.UnauthorizedError(info.message);
    } else {
      try {
        ctx.state.result = {
          data: {
```

```
            user,
          },
        };
      } catch (e) {
        throw new HTTPErrors.UnauthorizedError('Unnable to
autorize.');
      }
    }
  })(ctx, next);

export default passport;
```

crud

```
import Datastore from 'nedb';

const db = {};
db.users = new Datastore({ filename: 'db/users.db', autoload: true
});

const TYPE = {
  USER: 'user',
  ROOM: 'room',
  PRESENTATION: 'presentation',
};

export const MODE = {
  MASTER: 'master',
  SLAVE: 'slave',
};

export const addUser = user =>
  new Promise((resolve, reject) => {
    db.users.insert({ ...user, _type: TYPE.USER }, (err, data) => {
      if (err) reject(err);
      resolve(data);
    });
  });

export const getUserById = id =>
  new Promise((resolve, reject) => {
    db.users.findOne({ _id: id, _type: TYPE.USER }, (err, data) => {
      if (err) reject(err);
      resolve(data);
    });
  });

export const getUserByLogin = login =>
```

```
  new Promise((resolve, reject) => {
    db.users.findOne({ login, _type: TYPE.USER }, (err, data) => {
      if (err) reject(err);
      resolve(data);
    });
  });

export const addPresentation = presentation =>
  new Promise((resolve, reject) => {
    db.users.insert({ ...presentation, _type: TYPE.PRESENTATION },
(err, data) => {
      if (err) reject(err);
      resolve(data);
    });
  });

export const getPresentationById = (id, mode) =>
  new Promise((resolve, reject) => {
    db.users.findOne({ id, mode, _type: TYPE.PRESENTATION }, (err,
data) => {
      if (err) reject(err);
      resolve(data);
    });
  });
```

apiResponse.js

```
import HTTPErrors from 'http-custom-errors';

export default async (ctx, next) => {
  try {
    await next();
    const { code = 200, data = null } = ctx.state.result || {};
    ctx.status = code;
    ctx.set('content-type', 'application/json');
    ctx.body = { data, status: data ? 'OK' : 'No Content' };
  } catch (err) {
    if (err instanceof HTTPErrors.HTTPError) {
      ctx.status = err.code || 500;
      ctx.body = { data: null, status: err.status || 'Error', msg:
err.message || '' };
    } else ctx.throw(500, err);
  }
};
```

presentations

```
export default ({ secret = null, id, body }) => `<!doctype html>
  <html>
    <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, user-scalable=no">

      <title>reveal.js</title>

      <link rel="stylesheet" href="../css/reset.css">
      <link rel="stylesheet" href="../css/reveal.css">
      <link rel="stylesheet" href="../css/theme/black.css">

      <!-- Theme used for syntax highlighting of code -->
      <link rel="stylesheet" href="../lib/css/monokai.css">
    </head>
    <body>
      <div class="reveal">
        <div class="slides">
          ${body}
        </div>
      </div>

      <script src="../js/reveal.js"></script>
      <script src="../js/socket.js"></script>
      <script>
        Reveal.initialize({
          multiplex: {
            secret: '${secret}',
            id: '${id}',
            url: 'https://reveal-js-multiplex-ccjbegmaii.now.sh'
          },
          dependencies: [
            { src: '../plugin/multiplex/${secret ? 'master' :
'client'}.js', async: true },
            { src: '../plugin/markdown/marked.js' },
            { src: '../plugin/markdown/markdown.js' },
            { src: '../plugin/highlight/highlight.js', async: true }
          ]
        });
      </script>
    </body>
  </html>
```